EECE 5639 Computer Vision I

Lecture 18

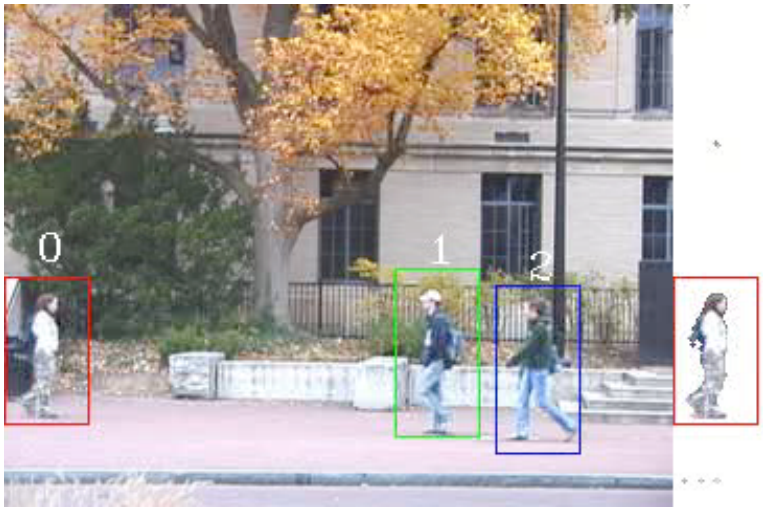
Motion: Motion Flow, Optical Flow estimation

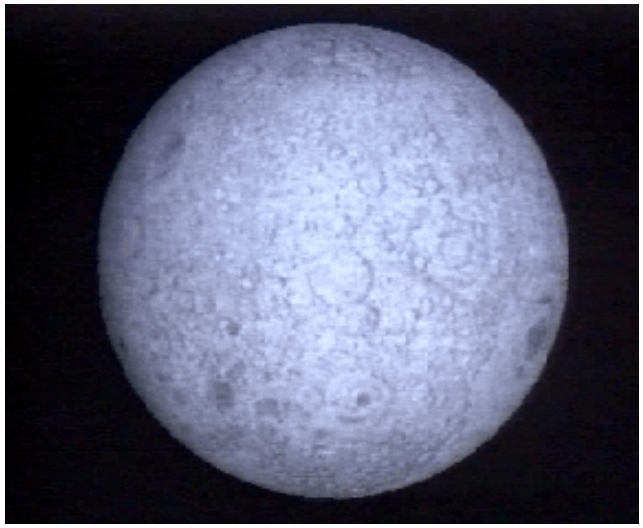
Hw 4 is out. Due March 29.

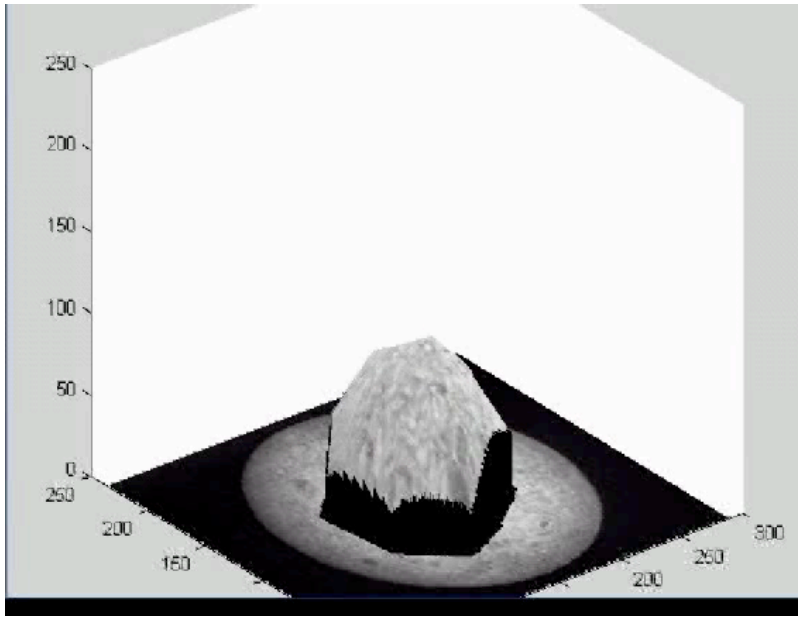
Next Class

Tracking

Motion


0
1
2




250
200
150
100
50
0
250
200
150
200
250
300

Process of images over time.

Image sequence:

An image sequence is a series of N images, or frames, acquired at discrete time instants $t_k=t_o+k\Delta t$, where Δt is a fixed time interval and $k=0,1,\ldots,N-1$

Assuming that illumination does not change:

Image changes are due to the RELATIVE MOTION between the scene and the camera.

There are 3 possibilities:

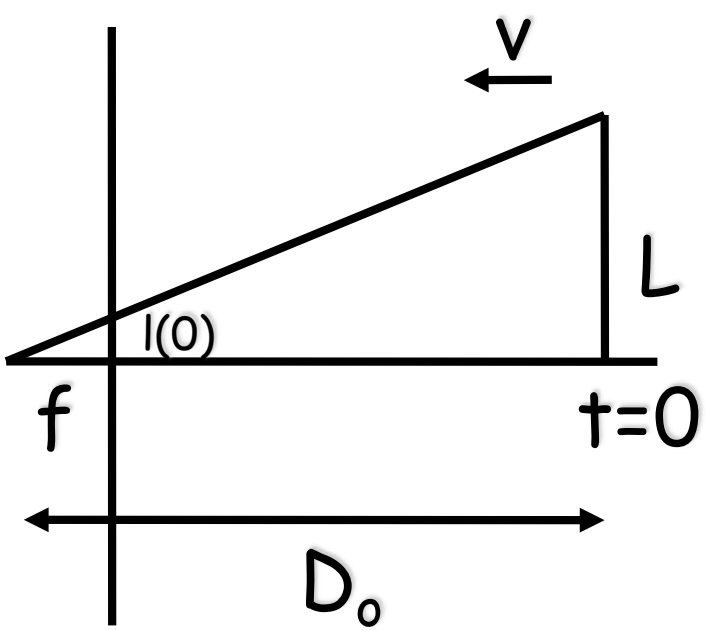
- Camera still, moving scene
- Moving camera, still scene
- Moving camera, moving scene

Visual Motion

Allows us to compute useful properties of the 3D world, with very little knowledge.

Example: Time to collision

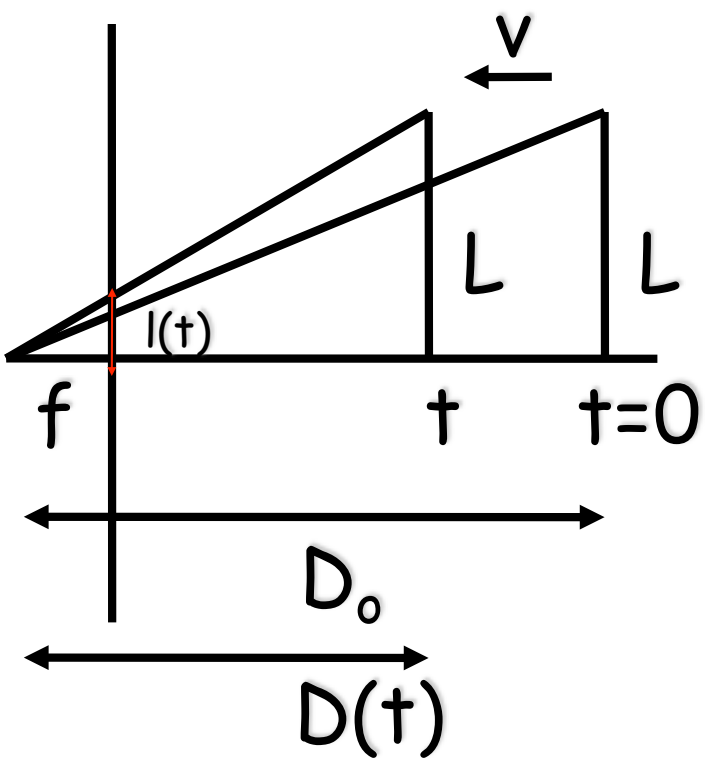
Time to Collision



An object of height L moves with constant velocity v:

- At time t=0 the object is at:
  - $D(0) = D_o$

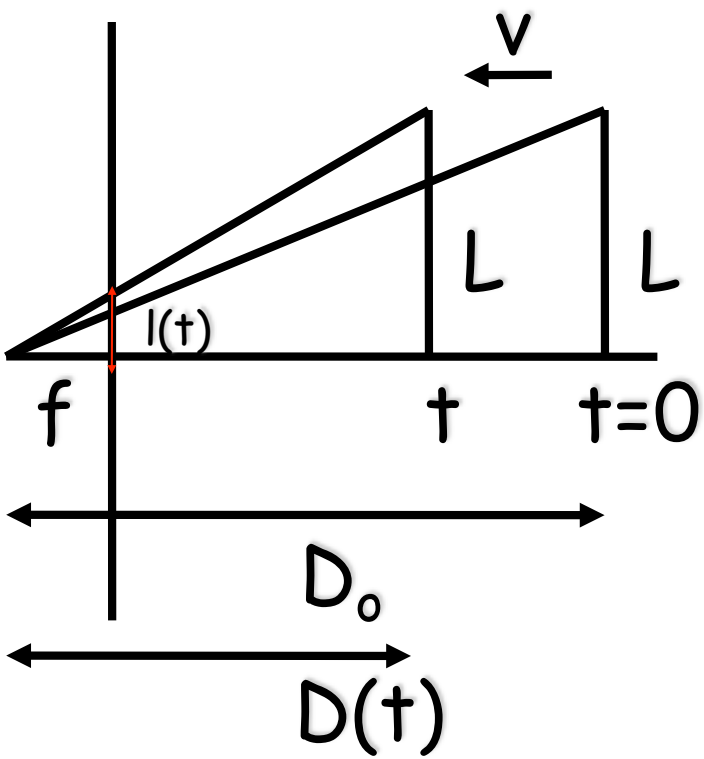
Time to Collision



An object of height L moves with constant velocity v:

- At time t=0 the object is at:
  - $D(0) = D_o$
- At time t it is at
  - $D(t) = D_o - vt$
- It will crash with the camera at time:
  - $D(\tau) = D_o - v\tau = 0$
  - $\tau = D_o/v$

Time to Collision



The image of the object has size l(t):

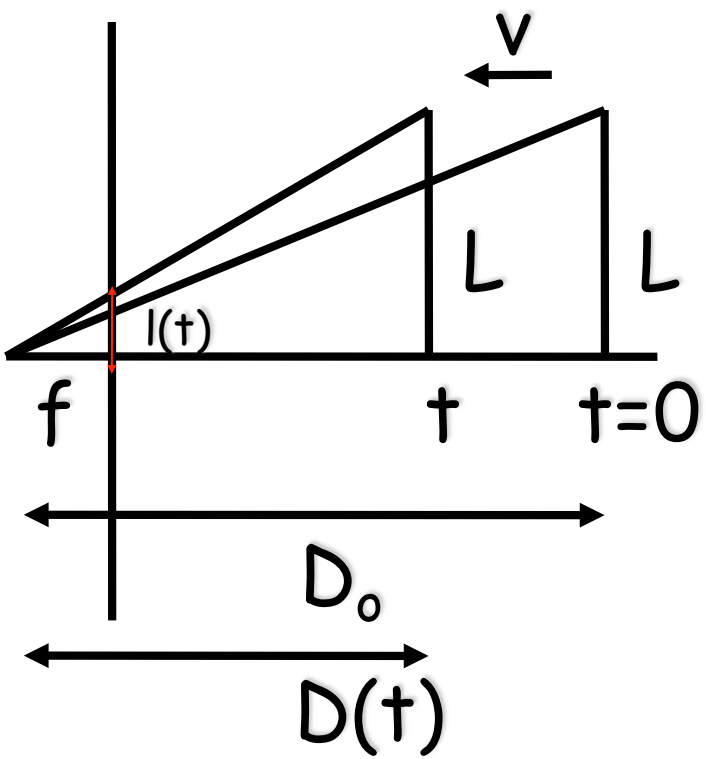
$$l(t) = \frac{fL}{D(t)}$$

Taking derivative wrt time:

$$l'(t) = \frac{dl(t)}{dt} = fL\frac{d(1/D(t))}{dt}$$

$$l'(t) = fL\frac{-1}{D^2(t)}\frac{d(D(t))}{dt}$$

Time to Collision



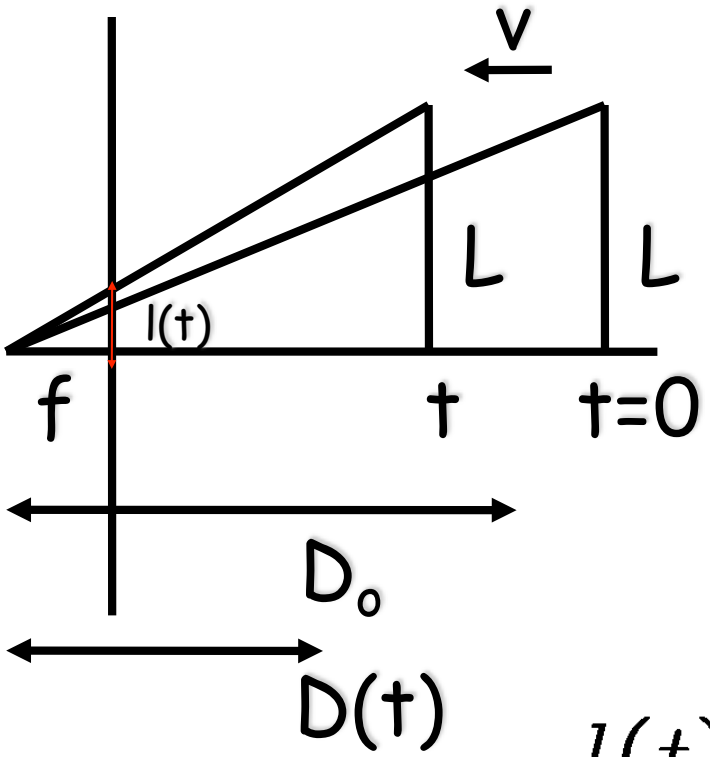
$$l'(t) = fL\frac{-1}{D^2(t)}\frac{d(D(t))}{dt}$$

$$D(t) = D_o - vt$$

$$\frac{d(D(t))}{dt} = -v$$

$$l'(t) = fL\frac{v}{D^2(t)}$$

Time to Collision



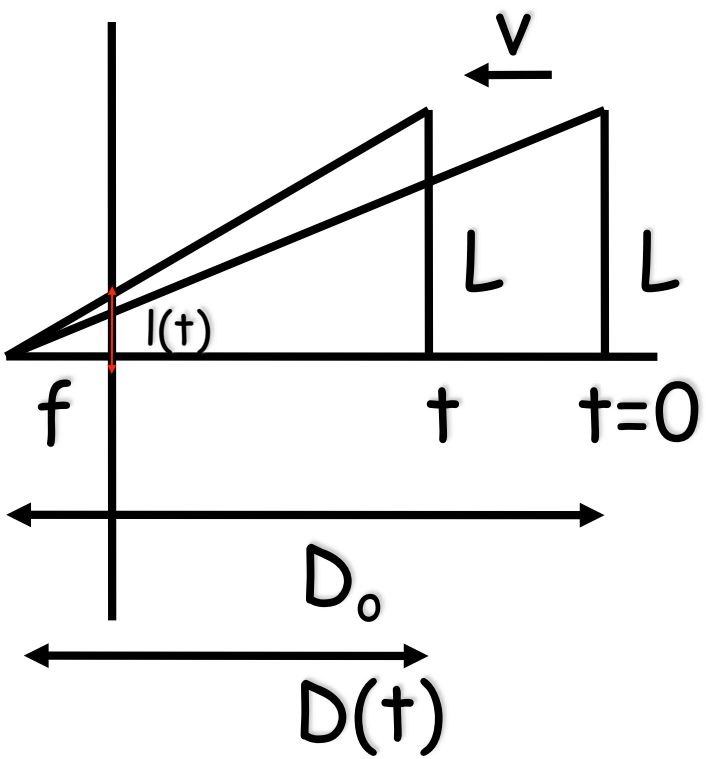
$$l'(t) = fL\frac{v}{D^2(t)}$$

$$l(t) = \frac{fL}{D(t)}$$

And their ratio is:

$$\frac{l(t)}{l'(t)} = \frac{fL}{D(t)}\frac{D^2(t)}{fLv} = \frac{D(t)}{v} = \tau$$

Time to Collision



$$l'(t) = fL\frac{v}{D^2(t)}$$

$$l(t) = \frac{fL}{D(t)}$$

Can be directly measured from image

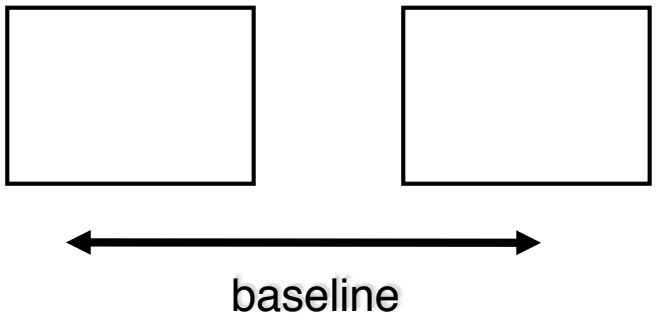
And time to collision:

$$\tau = \frac{l(t)}{l'(t)}$$

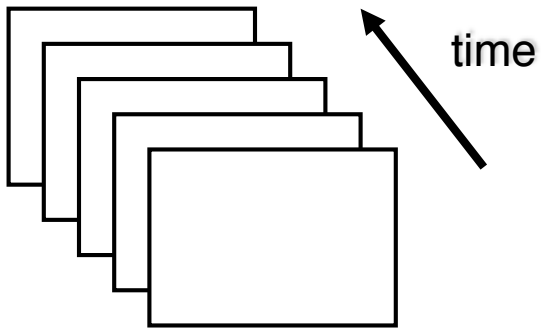
Can be found, without knowing L or D_o or v !!

Comparison between Motion Analysis and Stereo

Stereo: Two or more frames



Motion: N frames



- Baseline is usually larger in stereo than in motion:
  - Motion disparities tend to be smaller
- Stereo images are taken at the same time:
  - Motion disparities can be due to scene motion
  - There can be more than 1 transf. btw frames

Motion Analysis Problems

Correspondence Problem

Track corresponding elements across frames

Reconstruction Problem

Given a number of corresponding elements, and camera parameters, what can we say about the 3D motion and structure of the observed scene?

Segmentation Problem

What are the regions of the image plane which correspond to *different* moving objects?

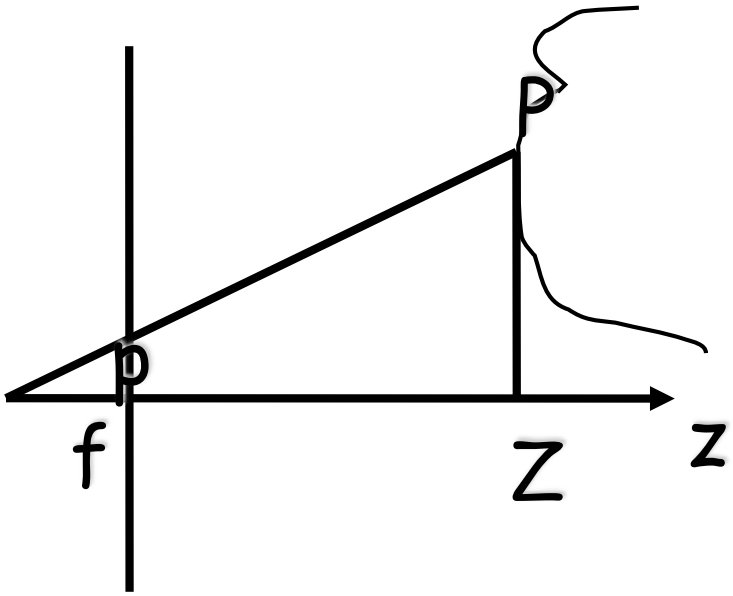
Motion Field (MF)

The MF assigns a velocity vector to each pixel in the image.

These velocities are INDUCED by the RELATIVE MOTION btw the camera and the 3D scene

The MF can be thought as the *projection* of the 3D velocities on the image plane.

Consider a 3D point P and its image:



$$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

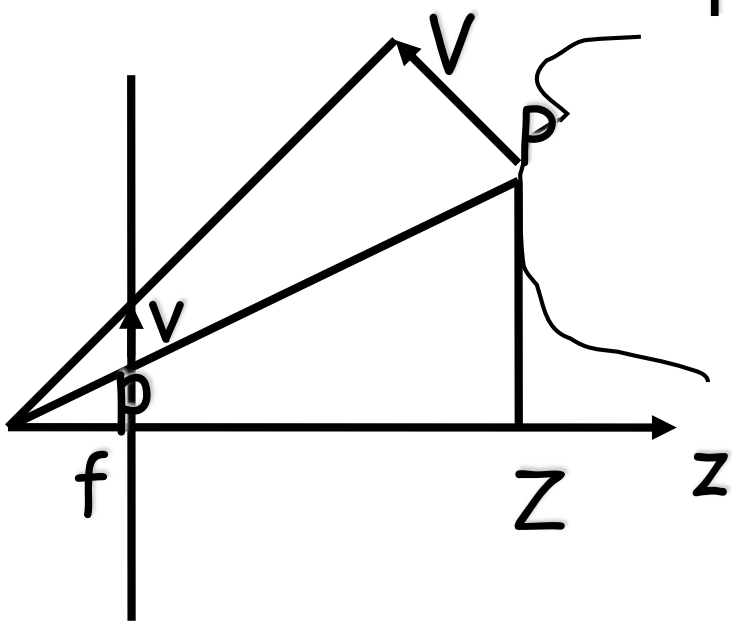
$$p = \begin{bmatrix} x \\ y \\ f \end{bmatrix}$$

Using the pinhole camera equation:

$$p = \frac{fP}{Z}$$

Let things move:

The relative velocity of P wrt camera:



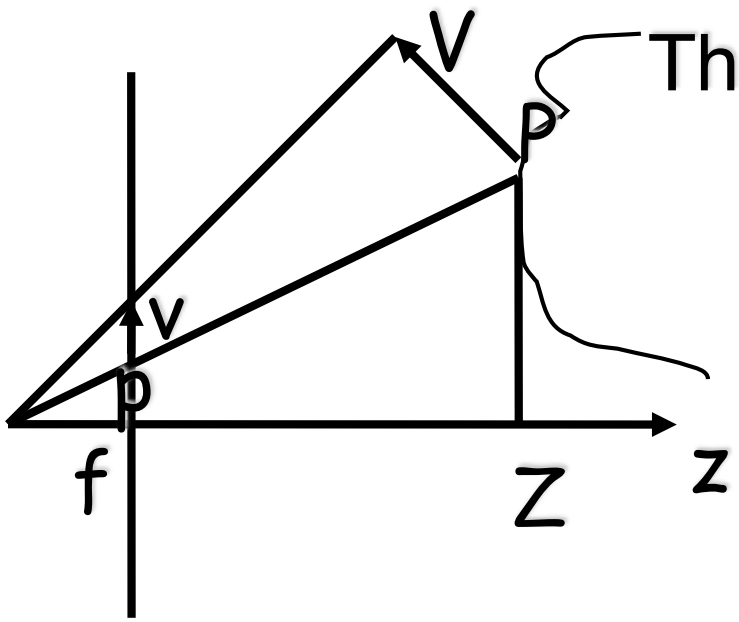
$$V = -T - \omega \times P$$

Translation velocity

Rotation angular velocity

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \qquad \omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

3D Relative Velocity:



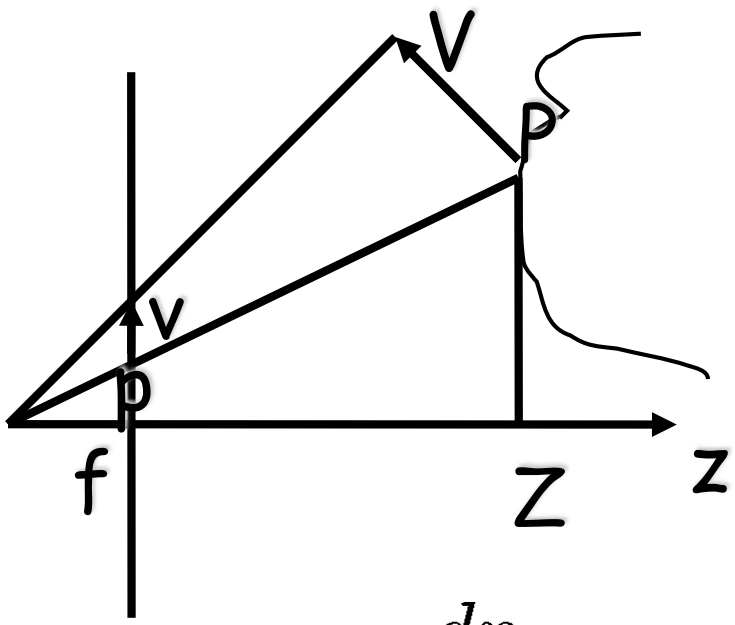
The relative velocity of P wrt camera:

$$V = -T - \omega \times P$$

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

$$\begin{aligned} V_x &= -T_x - \omega_y Z + \omega_z Y \\ V_y &= -T_y - \omega_z X + \omega_x Z \\ V_z &= -T_z - \omega_x Y + \omega_y X \end{aligned}$$

Motion Field: the velocity of p



$$p = \frac{fP}{Z}$$

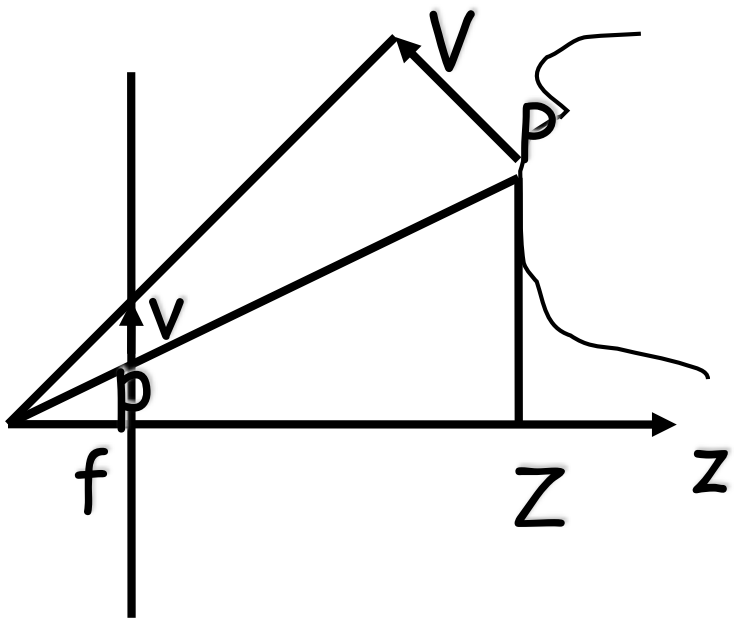
Taking derivative wrt time:

$$\frac{dp}{dt} = v = \frac{d\frac{fP}{Z}}{dt}$$

$$\frac{dp}{dt} = v = \frac{f}{Z^2}\left[\frac{dP}{dt}.Z - P.\frac{dZ}{dt}\right]$$

$$\frac{dp}{dt} = v = \frac{f}{Z^2}\left[V.Z - P.V_z\right]$$

Motion Field: the velocity of p

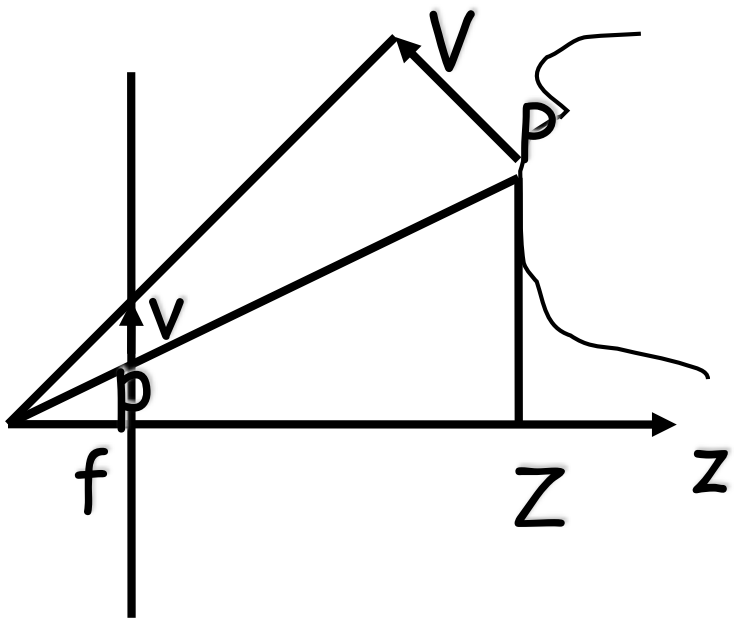


$$\frac{dp}{dt} = v = \frac{f}{Z^2}\left[V.Z - P.V_z\right]$$

$$p = \frac{fP}{Z} \qquad P = \frac{pZ}{f}$$

$$v = f\frac{V}{Z} - p\frac{V_z}{Z}$$

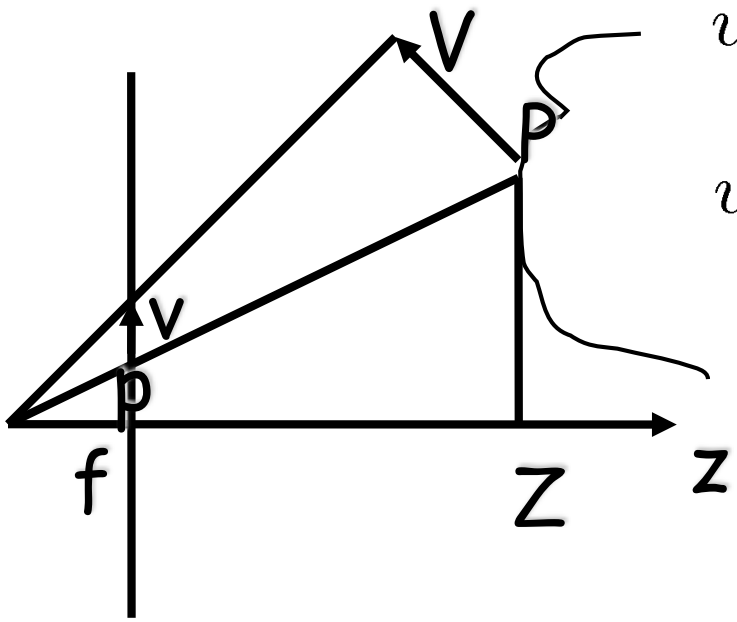
Motion Field: the velocity of p



$$v = f\frac{V}{Z} - p\frac{V_z}{Z}$$

$$\begin{aligned} v_x &= f\frac{V_x}{Z} - x\frac{V_z}{Z} \\ v_y &= f\frac{V_y}{Z} - y\frac{V_z}{Z} \\ v_z &= f\frac{V_z}{Z} - f\frac{V_z}{Z} = 0 \end{aligned}$$

Motion Field: the velocity of p



$$v_x = f\frac{V_x}{Z} - x\frac{V_z}{Z}$$

$$v_y = f\frac{V_y}{Z} - y\frac{V_z}{Z}$$

$$V_x = -T_x - \omega_y Z + \omega_z Y$$

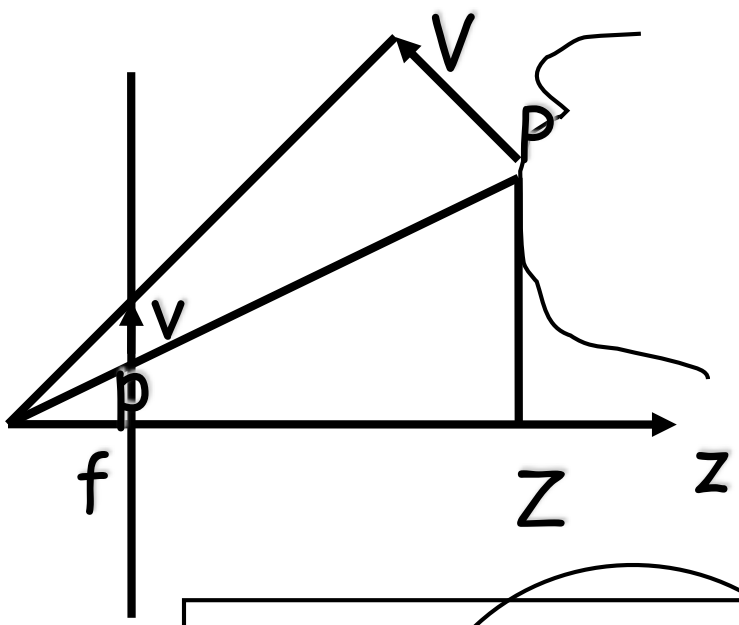
$$V_y = -T_y - \omega_z X + \omega_x Z$$

$$V_z = -T_z - \omega_x Y + \omega_y X$$

$$v_x = \frac{T_z x - T_x f}{Z} - \omega_y f + \omega_z y + \frac{\omega_x xy}{f} - \frac{\omega_y x^2}{f}$$

$$v_y = \frac{T_z y - T_y f}{Z} + \omega_x f - \omega_z x - \frac{\omega_y xy}{f} + \frac{\omega_x y^2}{f}$$

Motion Field: the velocity of p

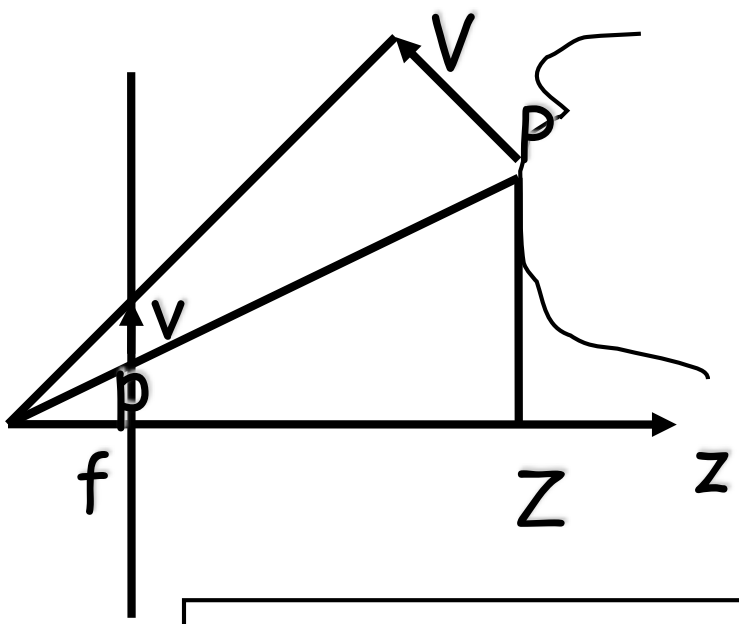


Translational component

$$v_x = \frac{T_z x - T_x f}{Z} - \omega_y f + \omega_z y + \frac{\omega_x xy}{f} - \frac{\omega_y x^2}{f}$$

$$v_y = \frac{T_z y - T_y f}{Z} + \omega_x f - \omega_z x - \frac{\omega_y xy}{f} + \frac{\omega_x y^2}{f}$$

Motion Field: the velocity of p



Rotational component

$$v_x = \frac{T_z x - T_x f}{Z} - \omega_y f + \omega_z y + \frac{\omega_x xy}{f} - \frac{\omega_y x^2}{f}$$

$$v_y = \frac{T_z y - T_y f}{Z} + \omega_x f - \omega_z x - \frac{\omega_y xy}{f} + \frac{\omega_x y^2}{f}$$

NOTE: The rotational component is independent of depth Z !

Special Case I: Pure Translation

$$\Rightarrow \quad v_x = \frac{T_z x - T_x f}{Z}$$

$$v_y = \frac{T_z y - T_y f}{Z}$$

Assume $T_z \neq 0$

Define:

$$p_o = \begin{bmatrix} x_o \\ y_o \\ f \end{bmatrix} = \begin{bmatrix} \frac{f T_x}{T_z} \\ \frac{f T_y}{T_z} \\ f \end{bmatrix}$$

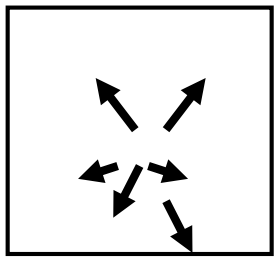
$$\Rightarrow \quad v_x = \frac{T_z x - T_z x_o}{Z} = (x - x_o)\frac{T_z}{Z}$$

$$v_y = \frac{T_z y - T_z y_o}{Z} = (y - y_o)\frac{T_z}{Z}$$

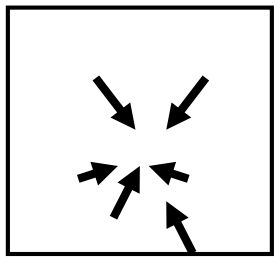
Special Case I: Pure Translation

$$v_x = (x - x_o)\frac{T_z}{Z}$$

$$v_y = (y - y_o)\frac{T_z}{Z}$$



$T_z > 0$



$T_z < 0$

The motion field in this case is RADIAL:

- It consists of vectors passing through $p_o = (x_o, y_o)$
- If:
  - $T_z > 0$, (camera moving towards object)
    - the vectors point away from p_o
    - p_o is the POINT OF EXPANSION
  - $T_z < 0$, (camera moving away from object)
    - the vectors point towards p_o
    - p_o is the POINT OF CONTRACTION

Special Case I: Pure Translation

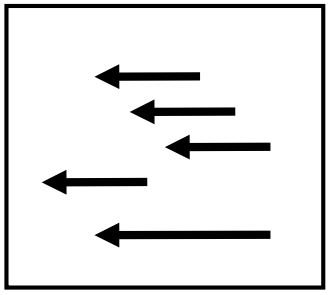
$$v_x = \frac{T_z x - T_x f}{Z}$$

$$v_y = \frac{T_z y - T_y f}{Z}$$

What if $T_z = 0$?

$$v_x = -f\frac{T_x}{Z}$$

$$v_y = -f\frac{T_y}{Z}$$



All motion field vectors are parallel to each other and inversely proportional to depth !

Pure Translation: Properties of the MF

If $T_z \neq 0$ the MF is RADIAL with all vectors pointing towards (or away from) a single point p_o.

If $T_z = 0$ the MF is PARALLEL.

The length of the MF vectors is inversely proportional to depth Z.

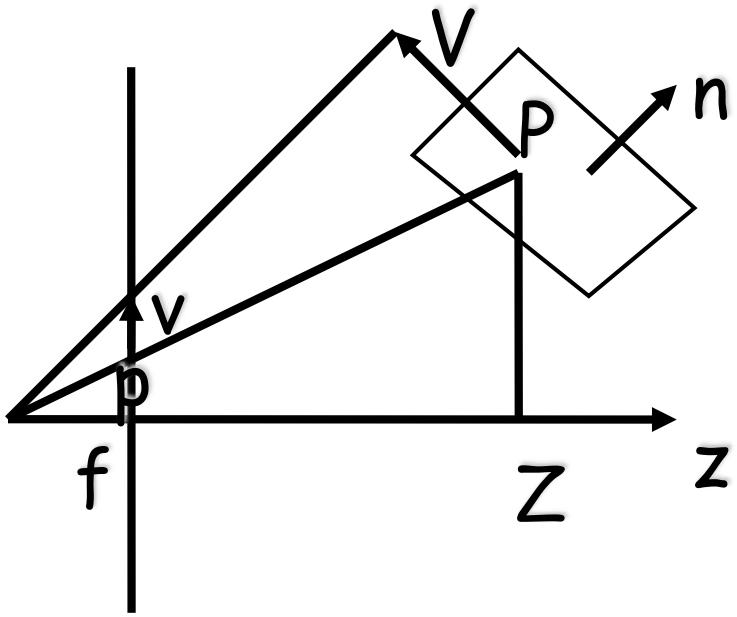
If $T_z \neq 0$ it is also directly proportional to the distance between p and p_o.

Pure Translation: Properties of the MF

p_o is the vanishing point of the direction of translation.

p_o is the intersection of the ray parallel to the translation vector and the image plane.

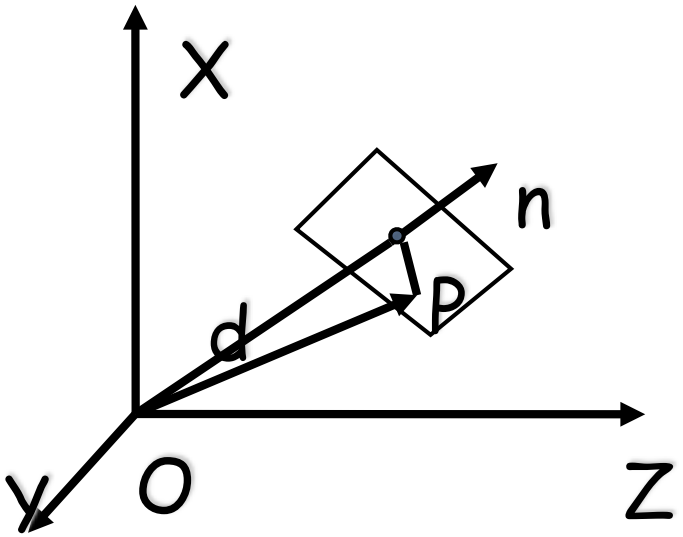
Special Case II: Moving Plane



Planar surfaces are common in man-made environments

Question: How does the MF of a moving plane look like?

Special Case II: Moving Plane



Points on the plane must satisfy the equation describing the plane.

Let

- n be the unit vector normal to the plane.
- d be the distance from the plane to the origin.
- NOTE: If the plane is moving wrt camera, n and d are functions of time.

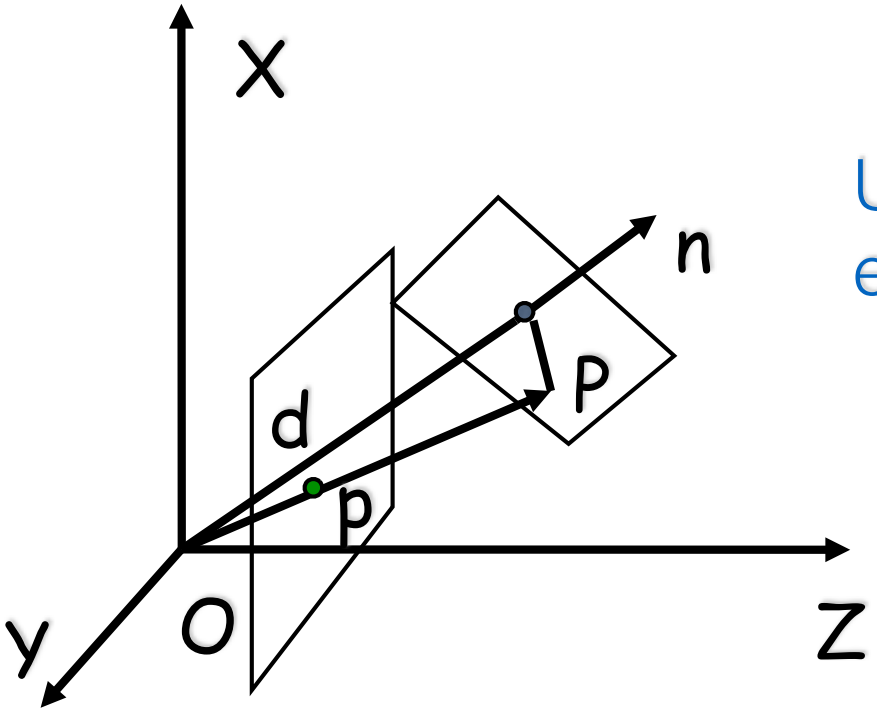
Then:

$$\mathbf{n}^T.\mathbf{P} = d$$

where $n = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$ $P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$

Special Case II: Moving Plane

Let $p = \begin{bmatrix} x \\ y \\ f \end{bmatrix}$ be the image of P



Using the pinhole projection equation:

$$p = \frac{fP}{Z} \Rightarrow P = \frac{pZ}{f}$$

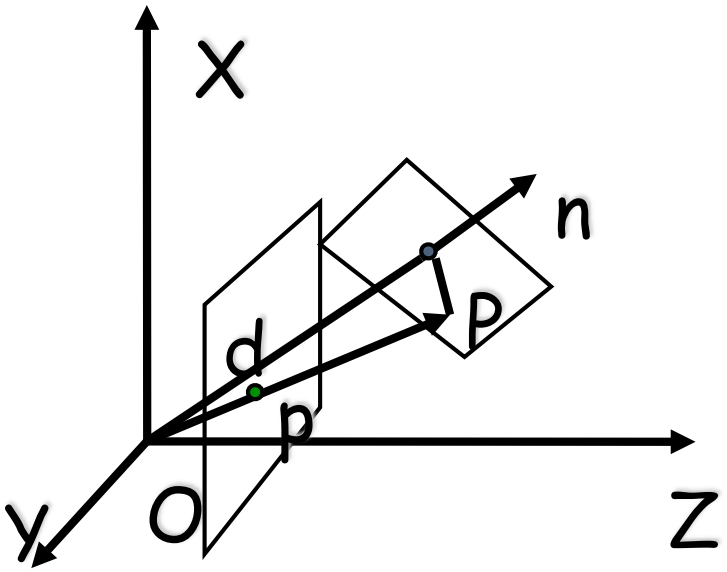
Using the plane equation:

$$\mathbf{n}^T.\mathbf{P} = d \Rightarrow \mathbf{n}^T.\mathbf{p}\frac{Z}{f} = d$$

Solving for Z:

$$\boxed{Z = \frac{fd}{n_x x + n_y y + n_z f}}$$

Special Case II: Moving Plane



Now consider the MF equations:

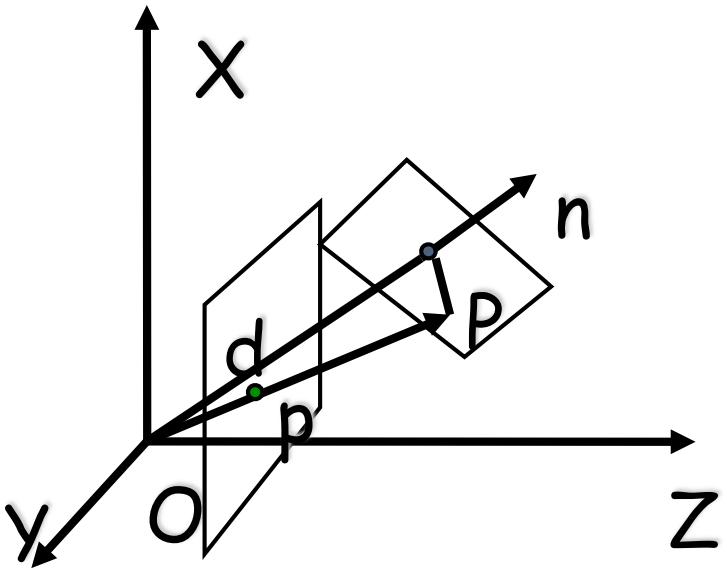
$$v_x = \frac{T_z x - T_x f}{Z} - \omega_y f + \omega_z y + \frac{\omega_x xy}{f} - \frac{\omega_y x^2}{f}$$

$$v_y = \frac{T_z y - T_y f}{Z} + \omega_x f - \omega_z x - \frac{\omega_y xy}{f} + \frac{\omega_x y^2}{f}$$

And Plug in:

$$Z = \frac{fd}{n_x x + n_y y + n_z f}$$

Special Case II: Moving Plane



The MF equations become:

$$v_x = \frac{1}{fd}(a_1x^2 + a_2xy + a_3fx + a_4fy + a_5f^2)$$

$$v_y = \frac{1}{fd}(a_1xy + a_2y^2 + a_6fy + a_7fx + a_8f^2)$$

where

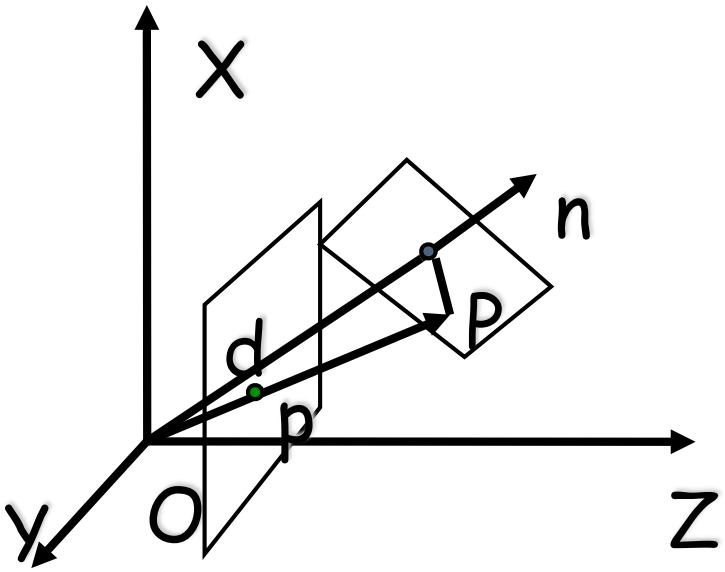
$$a_1 = -d\omega_y + T_zn_x \qquad a_2 = d\omega_x + T_zn_y$$

$$a_3 = T_zn_z - T_xn_x \qquad a_4 = d\omega_z - T_xn_y$$

$$a_5 = -d\omega_y - T_xn_z \qquad a_6 = T_zn_z - T_yn_y$$

$$a_7 = -d\omega_z - T_yn_x \qquad a_8 = d\omega_x - T_yn_z$$

Special Case II: Moving Plane



MF equations:

$$v_x = \frac{1}{fd}(a_1x^2 + a_2xy + a_3fx + a_4fy + a_5f^2)$$

$$v_y = \frac{1}{fd}(a_1xy + a_2y^2 + a_6fy + a_7fx + a_8f^2)$$

Q: What is the significance of this?

A: The MF vectors are given by low order (second) polynomials.

- Their coeffs. a_1 to a_8 (only 8, not 10 !) are functions of n, d, T and ω.
- The same coeffs. can be obtained with a different plane and relative velocity.

Moving Plane: Properties of the MF

The MF of a *planar* surface is at any time a *quadratic function* of the image coordinates.

A plane $n^T P=d$ moving with velocity $V=-T-\omega x P$ has the same MF than

a plane with normal $n'=T/|T|$, distance d and moving with velocity

$V'=|T|n -(\omega + n x T/d)x P$

Estimating the MF

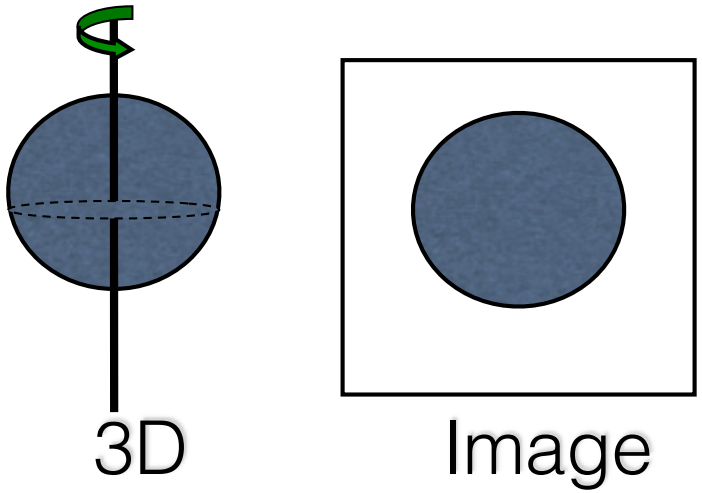
Optical Flow

MF Estimation

We will use the apparent motion of brightness patterns observed in an image sequence. This motion is called OPTICAL FLOW (OF)

MF ≠ OF

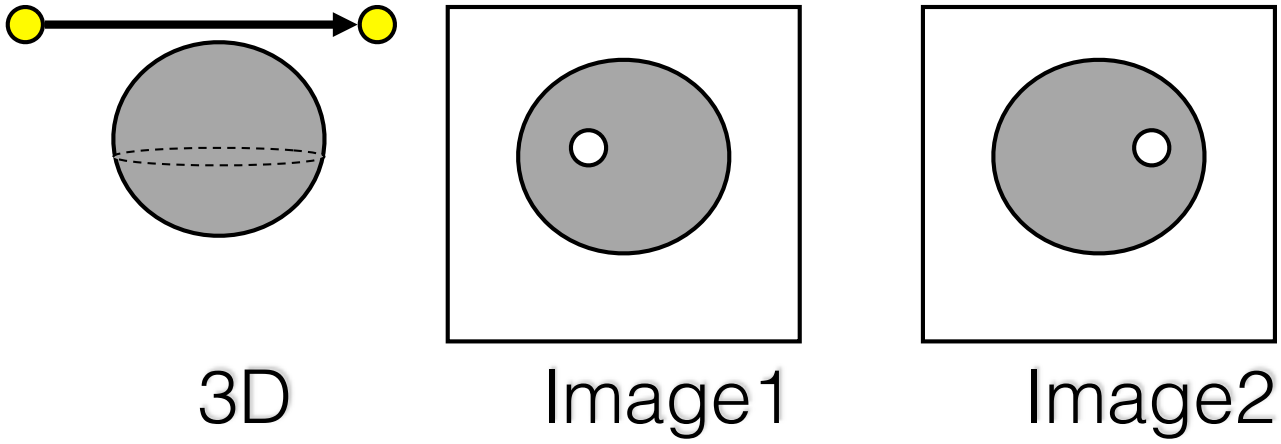
Consider a smooth, lambertian, uniform sphere rotating around a diameter, in front of a camera:



- MF ≠ 0 since the points on the sphere are moving
- OF = 0 since there are no moving patterns in the images

MF ≠ OF

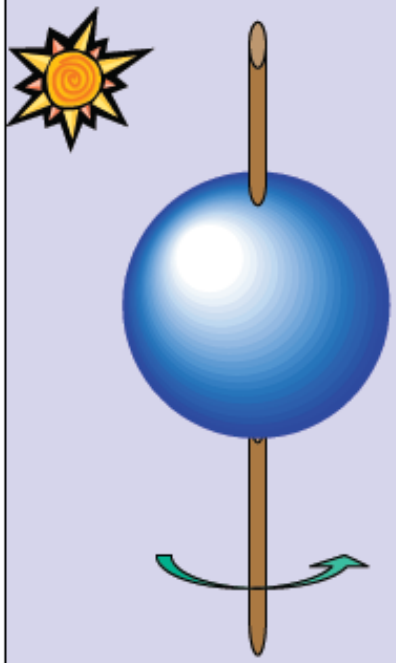
Consider a still, smooth, specular, uniform sphere, in front of a stationary camera and a moving light source:



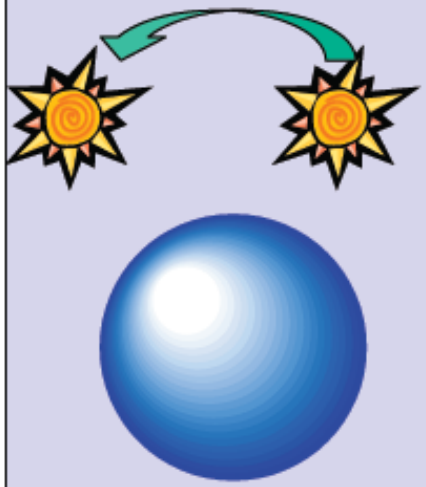
- MF = 0 since the points on the sphere are not moving
- OF ≠ 0 since there is a moving pattern in the images

MF ≠ OF

The screen is stationary yet displays motion



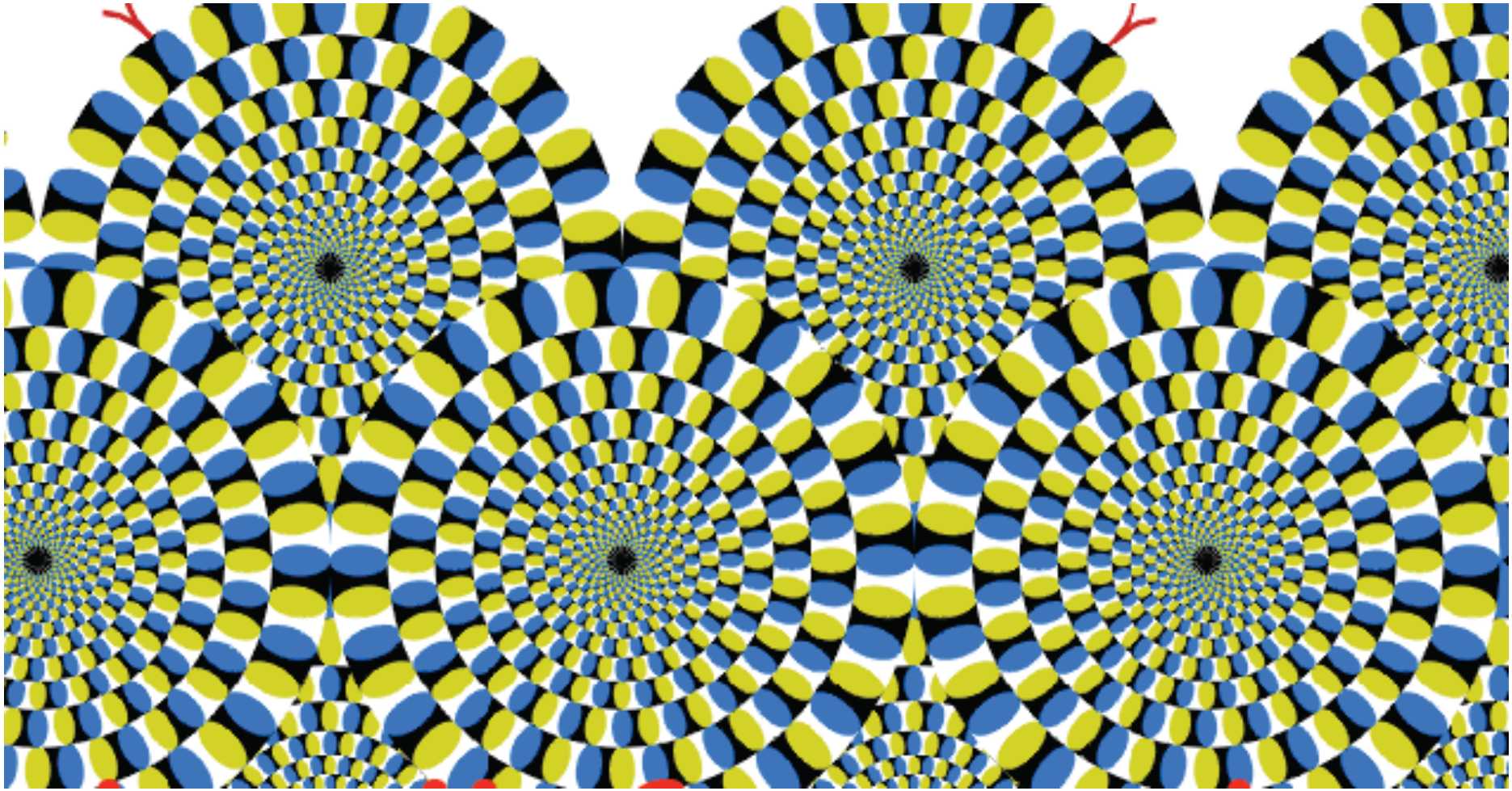
Homogeneous objects generate zero optical flow.



Fixed sphere. Changing light source.

Non-rigid texture motion

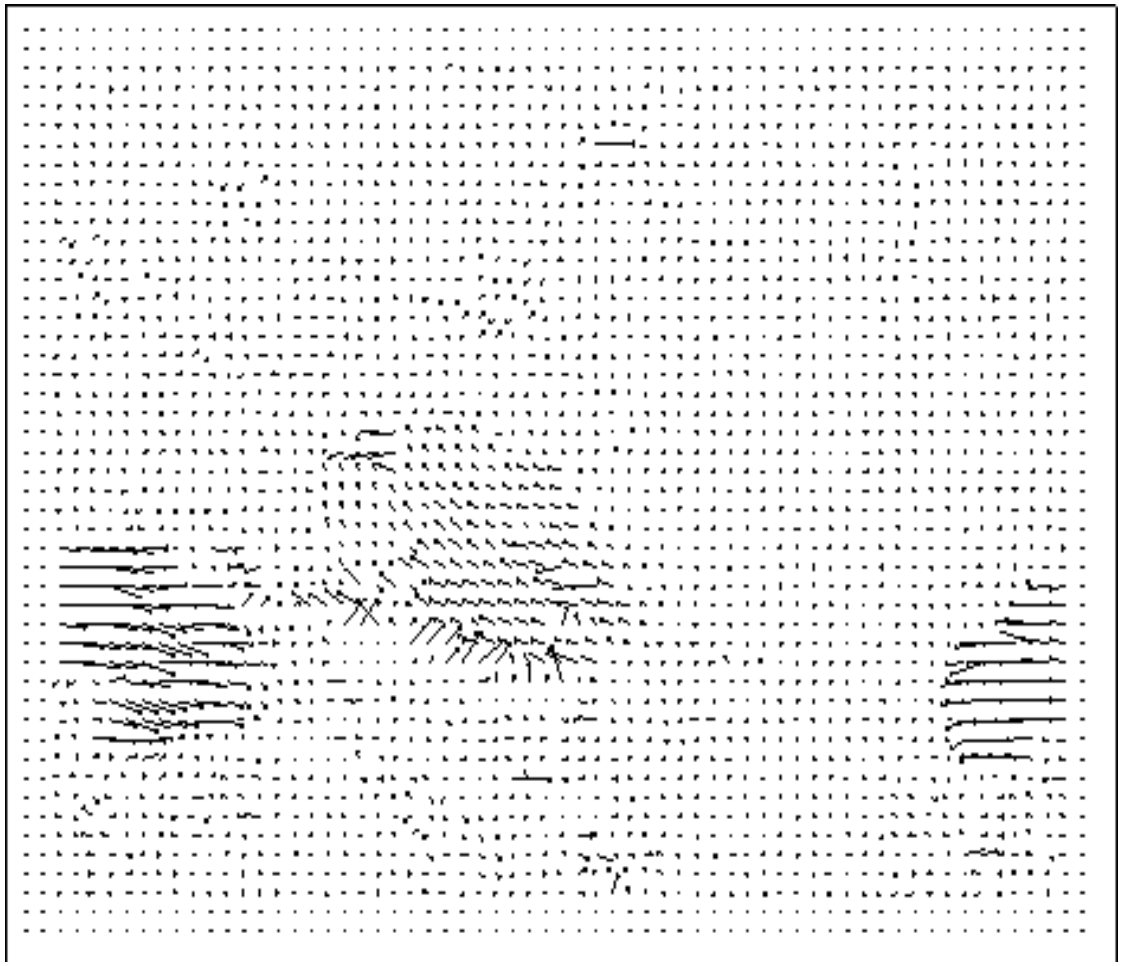
MF ≠ OF



Approximation of the MF

Never the less, keeping in mind that MF ≠ OF, we will assume that the apparent brightness of moving objects remains constant and hence we will estimate OF instead (since MF cannot really be observed!)

Optical Flow Computation



Brightness Constancy Equation

Let P be a moving point in 3D:

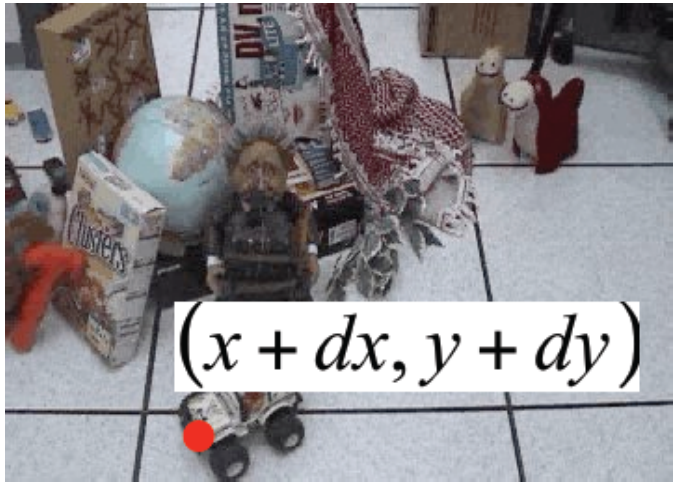
- At time t, P has coords (X(t),Y(t),Z(t))
- Let p=(x(t),y(t)) be the coords. of its image at time t.
- Let E(x(t),y(t),t) be the brightness at p at time t.

Brightness Constancy Assumption:

- As P moves over time, E(x(t),y(t),t) remains constant.

Brightness Constancy Equation

time t



time t + dt

$$E(x, y, t) = E(x + dx, y + dy, t + dt)$$

Brightness Constancy Equation

$$E(x(t), y(t), t) = Constant$$

Taking derivative wrt time:

$$\frac{dE(x(t), y(t), t)}{dt} = 0$$

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

Brightness Constancy Equation

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

Let

$$\nabla E = \begin{bmatrix} \frac{\partial E}{\partial x} \\ \frac{\partial E}{\partial y} \end{bmatrix}$$

(Frame spatial gradient)

$$v = \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix}$$

(optical flow)

$$E_t = \frac{\partial E}{\partial t}$$

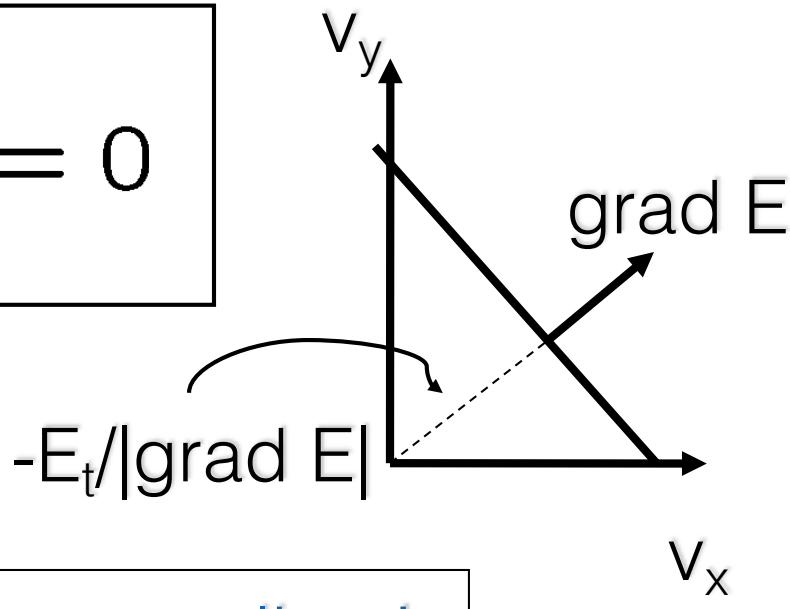
(derivative across frames)

Brightness Constancy Equation

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

Becomes:

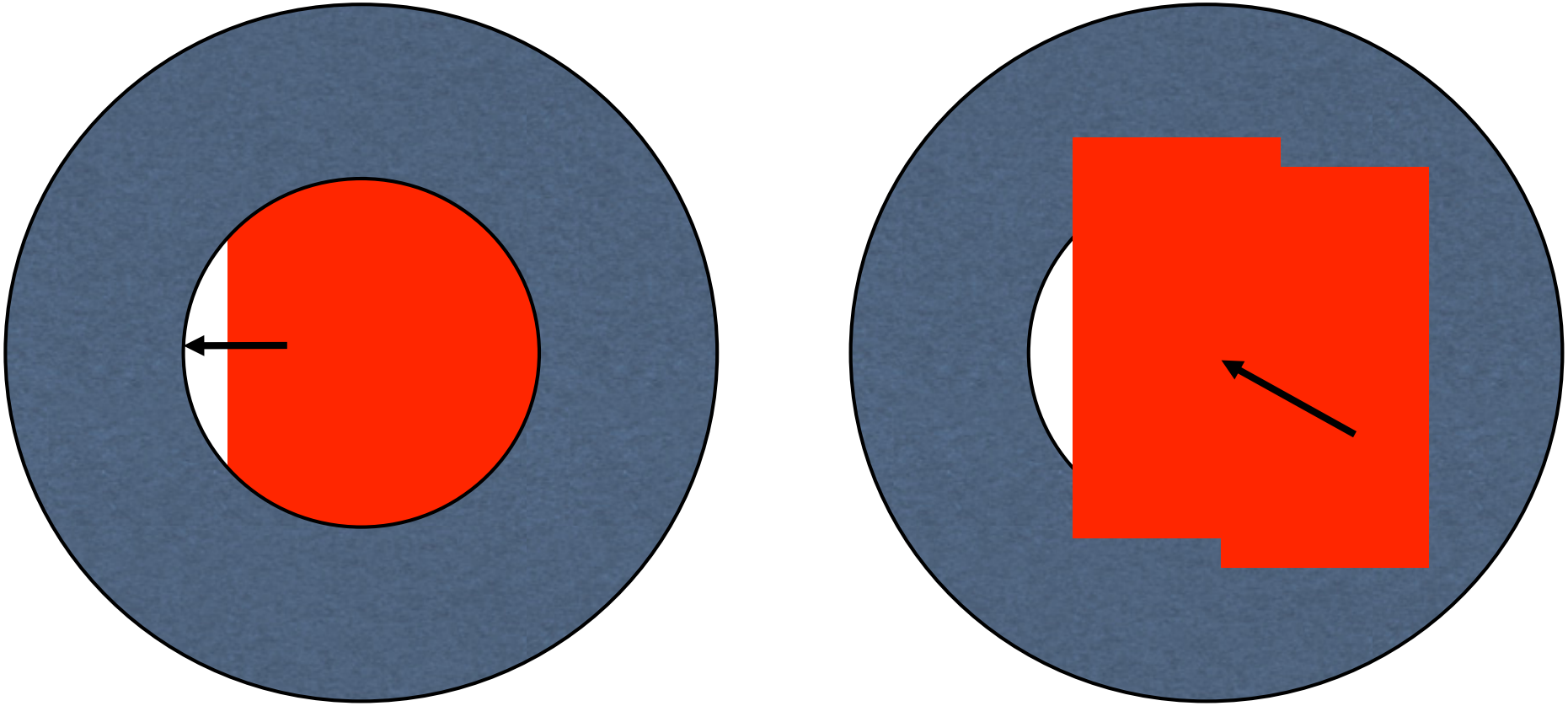
$$(\nabla E)^T . v + E_t = 0$$



The OF is CONSTRAINED to be on a line !

Problem: Two unknowns, one equation.

The aperture problem



The Image Brightness Constancy Assumption only provides the OF component in the direction of the spatial image gradient

Estimating OF

The brightness constancy assumption is not enough to compute OF:

- it only provides its normal component.
- Need some other assumption or knowledge.

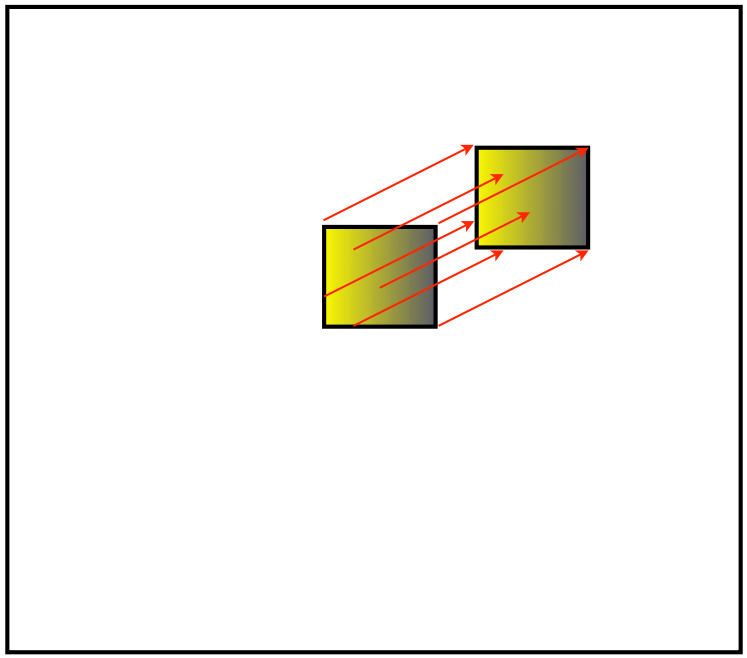
Algorithms computing OF are of two types:

- Differential Techniques
  - Based on spatial and temporal variations of the image brightness at all pixels.
  - Used to compute DENSE OF.
- Matching Techniques
  - Similar to stereo feature matching, compute disparities.
  - Used to compute SPARSE OF.

A differential technique

Local Smoothness Constraint: Lucas-Kanade (1984)

Assume constant OF in a small neighborhood:



A Differential OF Algorithm

Assumptions:

Brightness Constancy: $$(\nabla E)^T v + E_t = \epsilon \sim 0$$

where ε accounts for noise and model errors

OF is constant in small patches:

Let Q be a small NxN patch (say N=5)

If the OF v is the same at each pixel p_i in Q:

$$[(\nabla E(p_i))^T v + E_t(p_i)]^2 = \epsilon^2 \sim 0$$

Summing over all pixels in Q:

$$\Psi(v) = \sum_{p_i \in Q} [(\nabla E(p_i))^T v + E_t(p_i)]^2$$

We need to find OF v s.t. Ψ(v) is as small as possible

This is a Least Square Error (LSE) Problem.

It can be solved by finding v s.t. :

$$\min_{v} \Psi(v) = \min_{v} \sum_{p_i \in Q} [(\nabla E(p_i))^T v + E_t(p_i)]^2$$

$$\begin{aligned}
(\nabla E(p_1))^T v &\sim -E_t(p_1) \\
(\nabla E(p_2))^T v &\sim -E_t(p_2) \\
&\vdots \\
(\nabla E(p_{N^2}))^T v &\sim -E_t(p_{N^2})
\end{aligned}$$

Expanding the equations:

$$
\begin{aligned}
(E_x(p_1)E_y(p_1)).(v_x v_y) &\sim -E_t(p_1) \\
(E_x(p_2)E_y(p_2)).(v_x v_y) &\sim -E_t(p_2) \\
&\vdots \\
(E_x(p_{N^2})E_y(p_{N^2})).(v_x v_y) &\sim -E_t(p_{N^2})
\end{aligned}
$$

Expanding the equations:

$$
\begin{aligned}
E_x(p_1)v_x + E_y(p_1)v_y &\sim -E_t(p_1) \\
E_x(p_2)v_x + E_y(p_2)v_y &\sim -E_t(p_2) \\
&\vdots \\
E_x(p_{N^2})v_x + E_y(p_{N^2})v_y &\sim -E_t(p_{N^2})
\end{aligned}
$$

This a system of N^2 equations with 2 unknowns: v_x and v_y

Usually, $N^2 >> 2$

Writing it in matrix format:

$$Av = b$$

where:

$$v = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

$$A = \begin{bmatrix} E_x(p_1) & E_y(p_1) \\ E_x(p_2) & E_y(p_2) \\ \vdots & \vdots \\ E_x(p_{N^2}) & E_y(p_{N^2}) \end{bmatrix}$$

$$b = \begin{bmatrix} -E_t(p_1) \\ -E_t(p_2) \\ \vdots \\ -E_t(p_{N^2}) \end{bmatrix}$$

Solving for v:

A is N^2x2, with $N^2 > 2$:

it is not invertible!

Multiply both sides of Av = b by A^T:

$$A^T A v = A^T b$$

A^TA is square (2x2):

$(A^TA)^{-1}$ exists if $\det(A^TA) \neq 0$

Assuming that $(A^TA)^{-1}$ does exists:

$$(A^T A)^{-1} (A^T A) v = (A^T A)^{-1} A^T b$$

$$v = (A^T A)^{-1} A^T b$$

Taking a closer look at (A^TA)

$$A = \begin{bmatrix} E_x(p_1) & E_y(p_1) \\ E_x(p_2) & E_y(p_2) \\ \vdots & \vdots \\ E_x(p_{N^2}) & E_y(p_{N^2}) \end{bmatrix}$$

$$A^T = \begin{bmatrix} E_x(p_1) & E_x(p_2) & \dots & E_x(p_{N^2}) \\ E_y(p_1) & E_y(p_2) & \dots & E_y(p_{N^2}) \end{bmatrix}$$

$$A^T A = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix}$$

This is a matrix used for corner detection!

Taking a closer look at (A^TA)

The matrix for corner detection:

$$A^T A = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix}$$

is singular (not invertible) when $\det(A^TA) = 0$

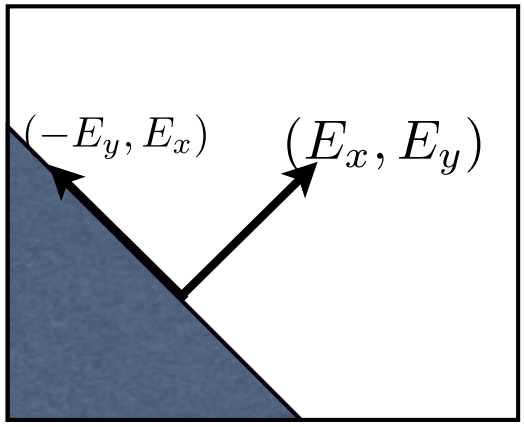
But $\det(A^TA) = \prod \lambda_i = 0$ -> one or both e.v. are 0

One e.v. = 0 -> no corner, just an edge
Two e.v. = 0 -> no corner, homogeneous region

} Aperture Problem !

Good Points to Estimate OF

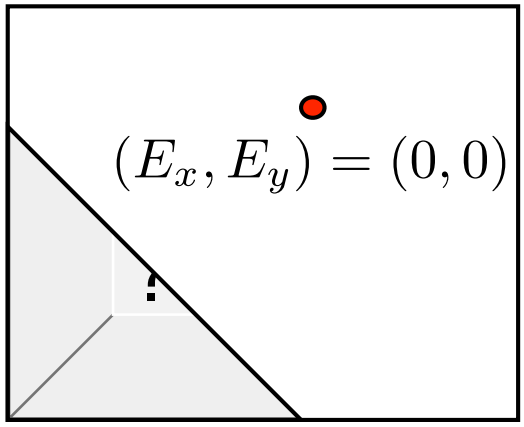
At edges, A^tA becomes singular



$$\begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix} \begin{bmatrix} -E_y \\ E_x \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} -E_y \\ E_x \end{bmatrix}$$ is an eigenvector with eigenvalue 0

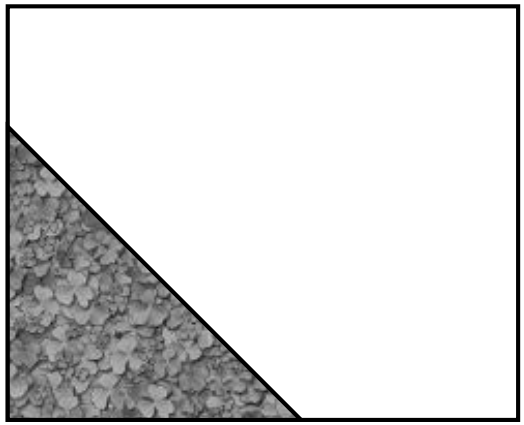
Good Points to Estimate OF



At homogeneous regions, A^tA becomes 0. (0 eigenvalues)

$$\begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Good Points to Estimate OF



Textured regions have two high eigenvalues: good OF estimation!

Algorithm CONSTANT_FLOW

Inputs:

A time varying sequence of n images: E_1,E_2,…,E_n

Parameters:

N: dimension of region Q

typically N=5

σ_s: stdev for Gaussian filter along x and y

typically 1.4 pixels-> w=2k+1=5σ_s =7 -> k = 3

discard borders of width k

σ_t: stdev for Gaussian filter along t

typically 1.4 frames-> w=2k+1=5σ_t =7 -> k = 3

discard first and last k frames

Algorithm CONSTANT_FLOW

For each image:

- Filter along x and y with a Gaussian filter
- Filter along t with a Gaussian filter
- Compute E_x,E_y and E_t

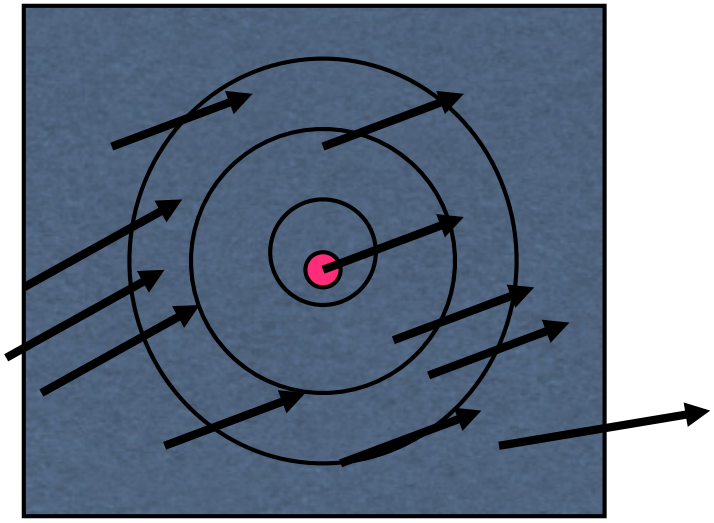
For each pixel of each image:

- Compute the matrix A and vector b
- Compute OF $v = (A^TA)^{-1}A^Tb$
- Output $v=(v_x,v_y)^T$ for each pixel

An improvement …

NOTE:

The assumption of constant OF is more likely to be wrong as we move away from the point of interest (the center point of Q)



Use weights to control the influence of the points: the farther from p, the less weight

Solving for v with weights:

Let W be a diagonal matrix with weights

Multiply both sides of Av = b by W:

$$W A v = W b$$

Multiply both sides of WAv = Wb by $(WA)^T$:

$$A^T WWA v = A^T WWb$$

$A^T W^2A$ is square (2x2):

$(A^TW^2A)^{-1}$ exists if $\det(A^TW^2A) \neq 0$

Assuming that $(A^TW^2A)^{-1}$ does exists:

$$(A^T W^2A)^{-1} (A^T W^2A) v = (A^T W^2A)^{-1} A^T W^2b$$

$$v = (A^T W^2A)^{-1} A^T W^2b$$

Feature-based Techniques

Feature Matching

- Estimates OF only at localized features
- Produces SPARSE OF mappings
- Two approaches:
  - Two-Frames methods
    - Similar to stereo
  - Multiple-Frames methods
    - Use previous frames to PREDICT location in next frame

Algorithm FEATURE_POINT_MATCHING

Inputs:

2 frames I_1 and I_2 and a set of point features in each frame (detected with the corner algorithm)

Parameters:

N: size of window Q

τ : threshold

Algorithm FEATURE_POINT_MATCHING

For each feature p in I_1,

let:

Q_1 be a NxN window centered at p

d be the unknown displacement between p and its corresponding feature in I_2

1. Set d = 0
2. Estimate $v = (A^TA)^{-1}A^Tb$ at p, using Q_1

Let d = d + v

Warp Q_1 to Q' using d. Let Q_2 be Q' in I_2

Compute SSD between Q_1 and Q'

1. IF SSD > τ, set Q_1 = Q' and go to 1, else exit.

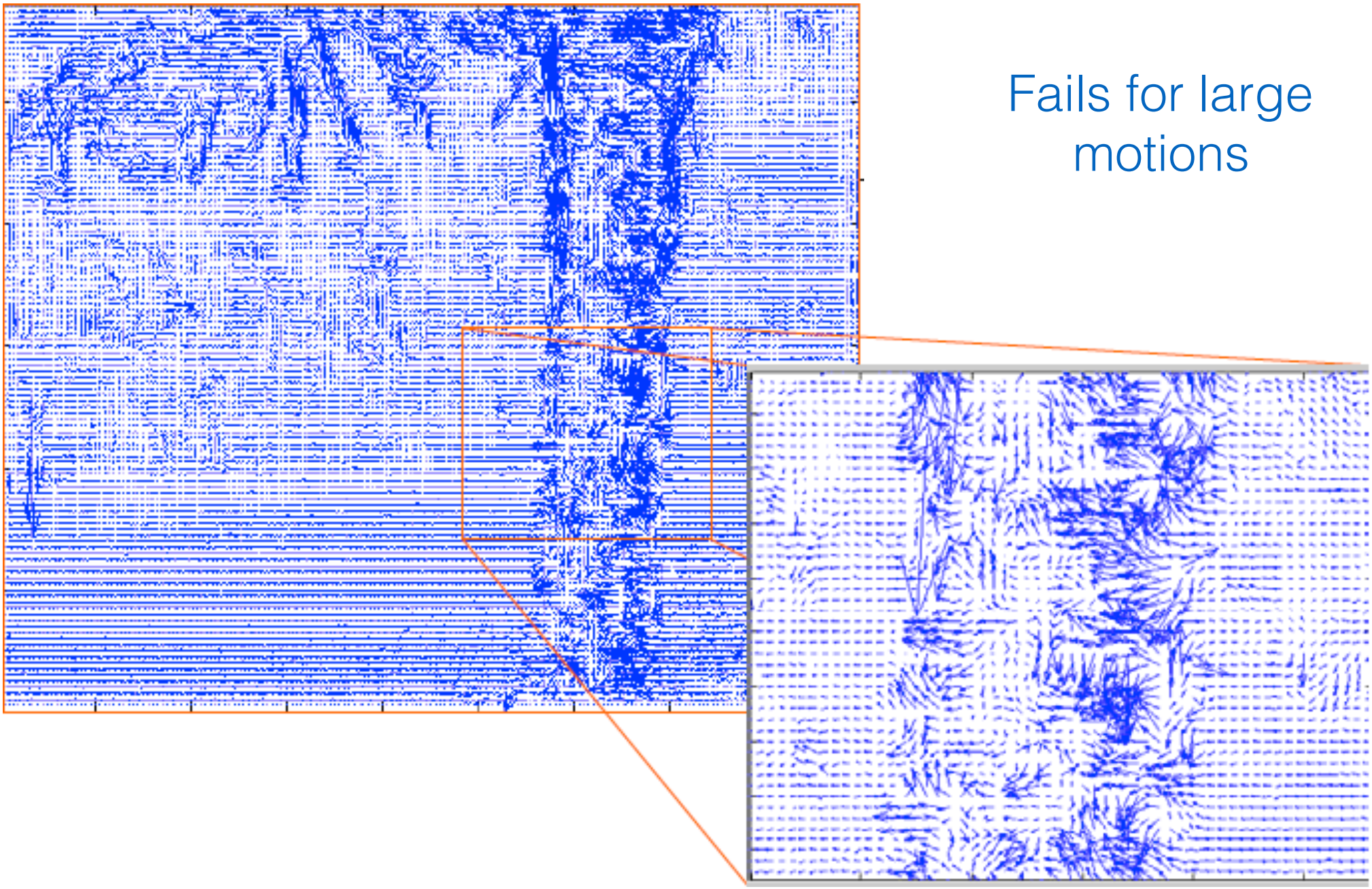
Large Motions

Frame 1

Frame 15

Frame 30

Large Motions



Fails for large motions

Multi-scale Gradient Descent

Traditional LK is just refining a position estimate:
We must be close to the right answer, and it only finds a local min.

We can increase range of LK methods by using a coarse to fine image pyramid, so at each level the refinement is small.

Large Motions

If object moves fast, the brightness changes quickly and small masks fail to estimate the spatiotemporal derivatives.

Pyramids can be used to compute large optical flow vectors.

Pyramids

Very useful for image representation

Built by using multiple copies at different resolutions

Each level in the pyramid is 1/4 size of the previous level (half columns, half rows)

The lowest level has the highest resolution

The highest level has the lowest resolution

Pyramid

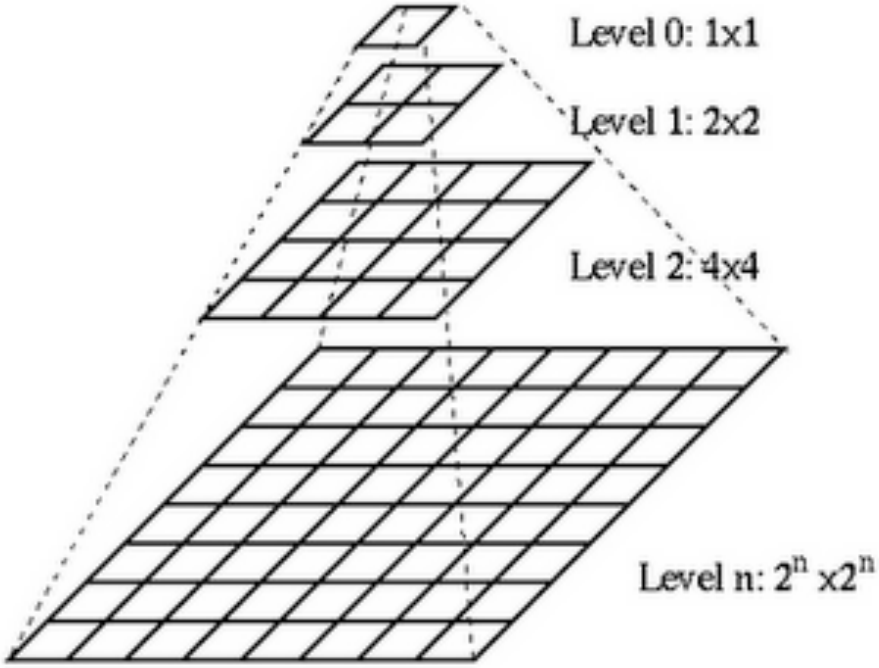
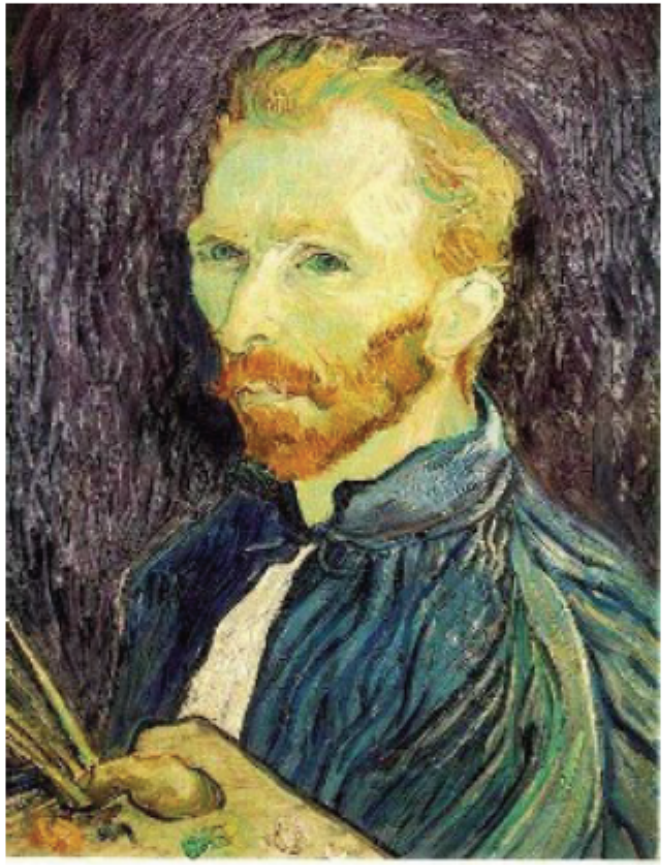


Image Sub-Sampling

Main idea: throw away every other row and column to create an image at 1/2 of the scale.

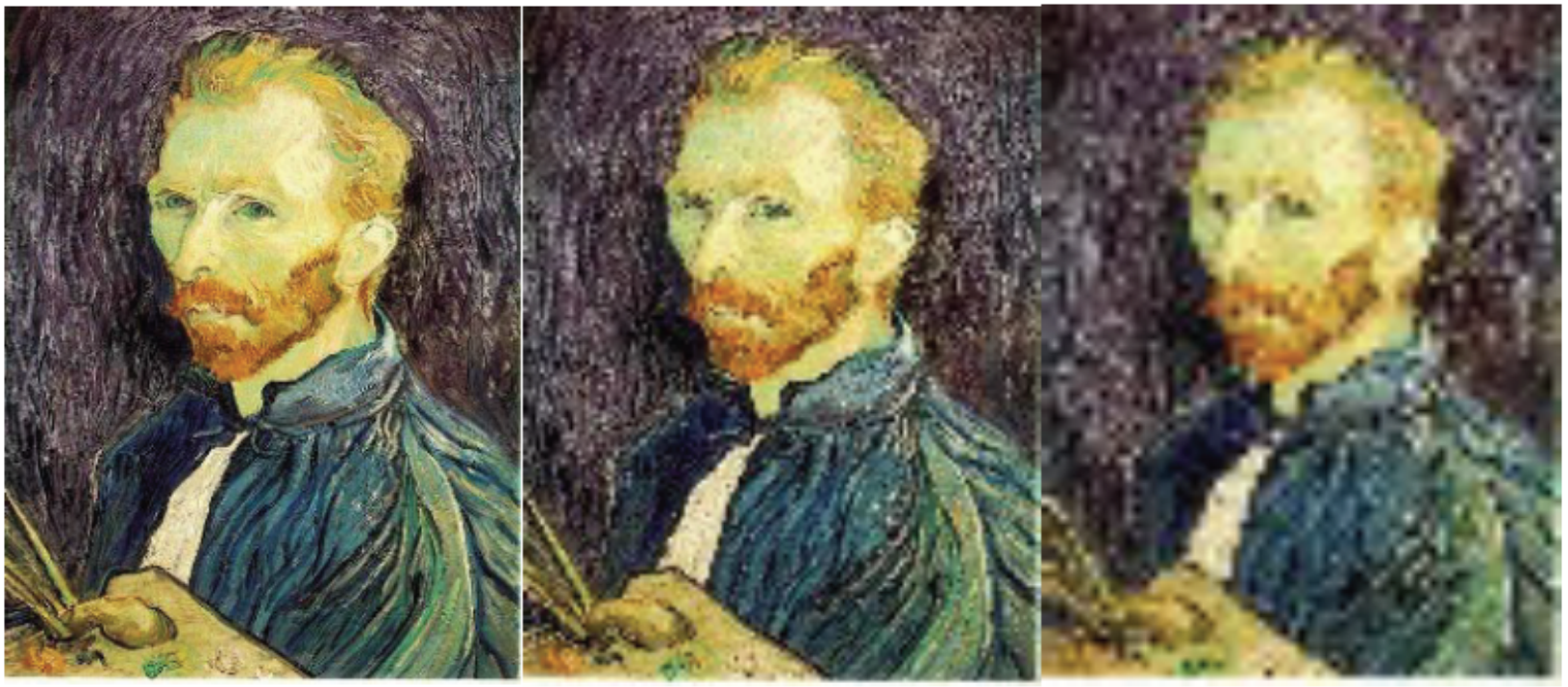


1/4

1/8

[Source: S. Seitz]

Zoom back …

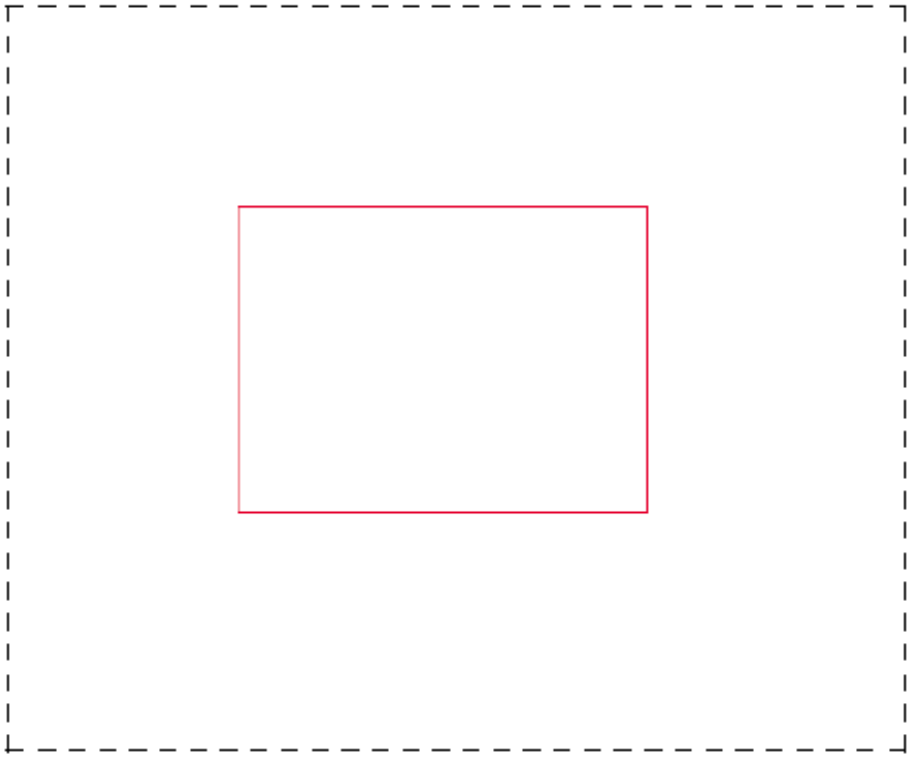


1/2 1/4 (2x zoom) 1/8 (4x zoom)

[Source: S. Seitz]

Why does it look so bad?

Even worse for synthetic images!



Let's sample every other row and column …

Even worse for synthetic images!

Where did the rectangle go???

Another fun example

What do you see?

[Source: Sanja Fidler]

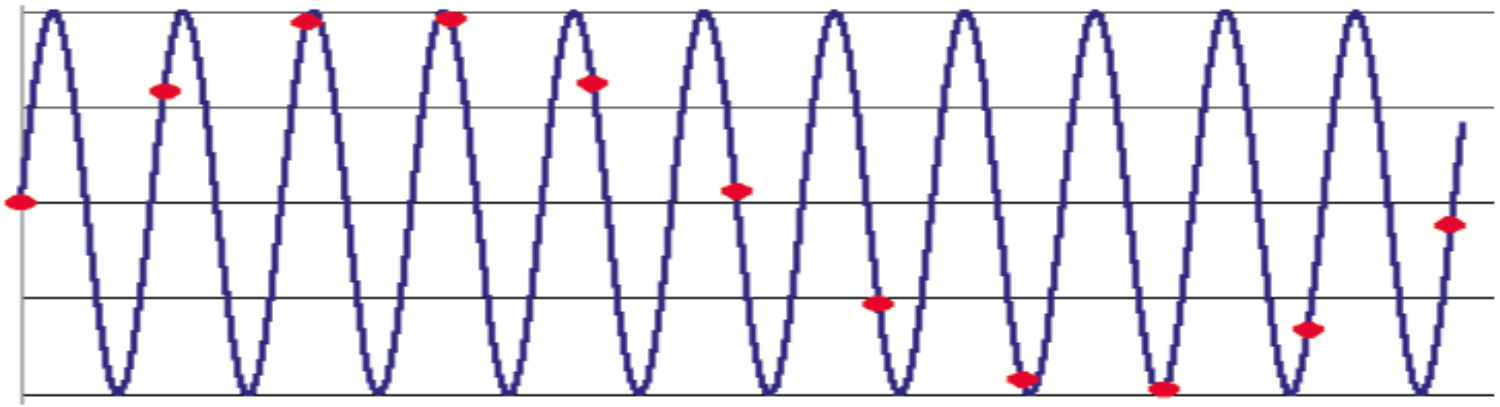
Subsample by columns

Where is the chicken?

[Source: Sanja Fidler]

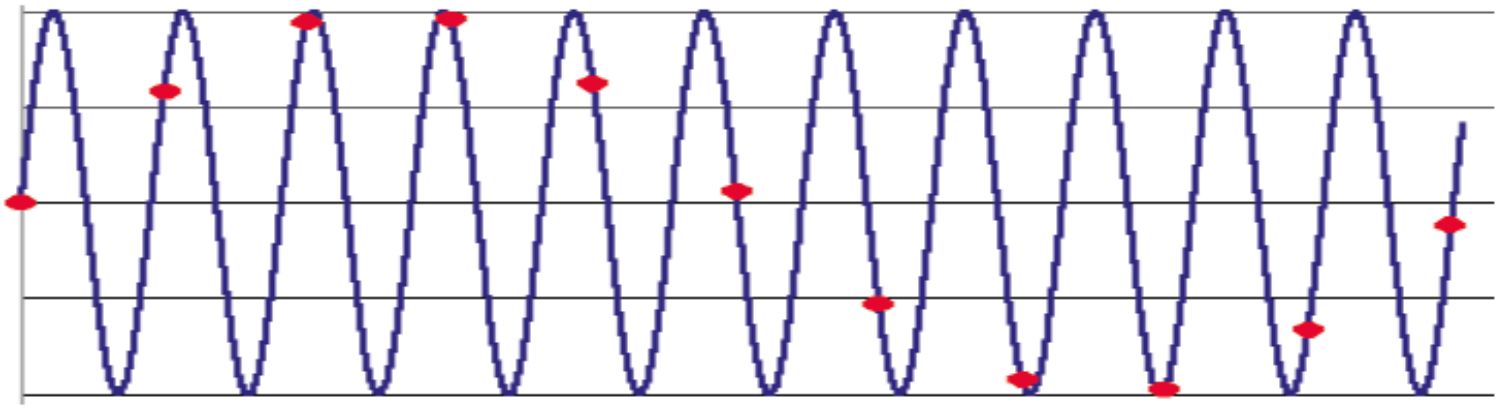
Aliasing

It happens when your sampling rate is not high enough to capture the amount of detail change.



Aliasing

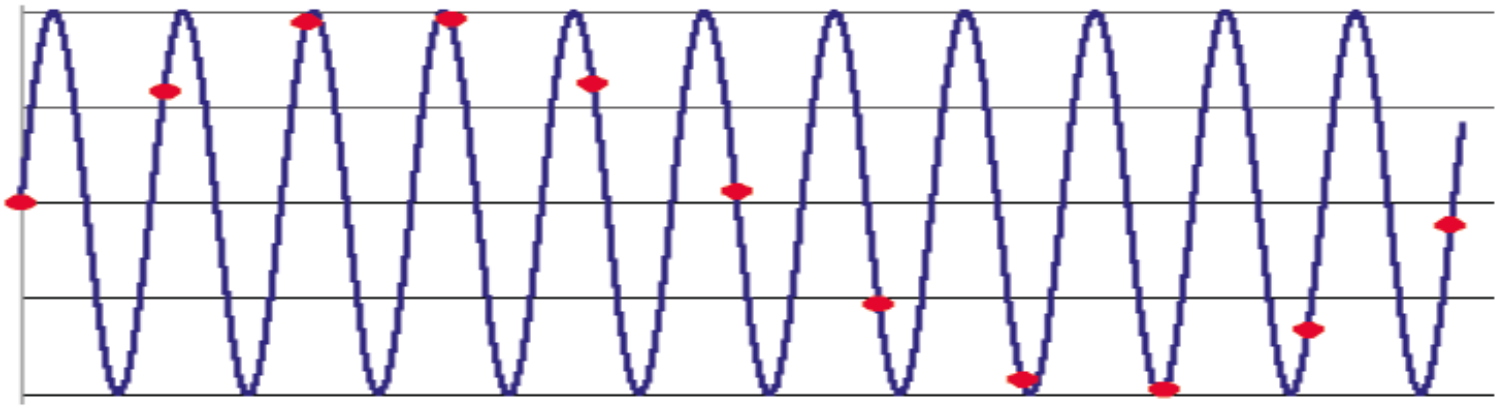
It happens when your sampling rate is not high enough to capture the amount of detail change.



To do sampling right, you need to understand the structure of your signal/image

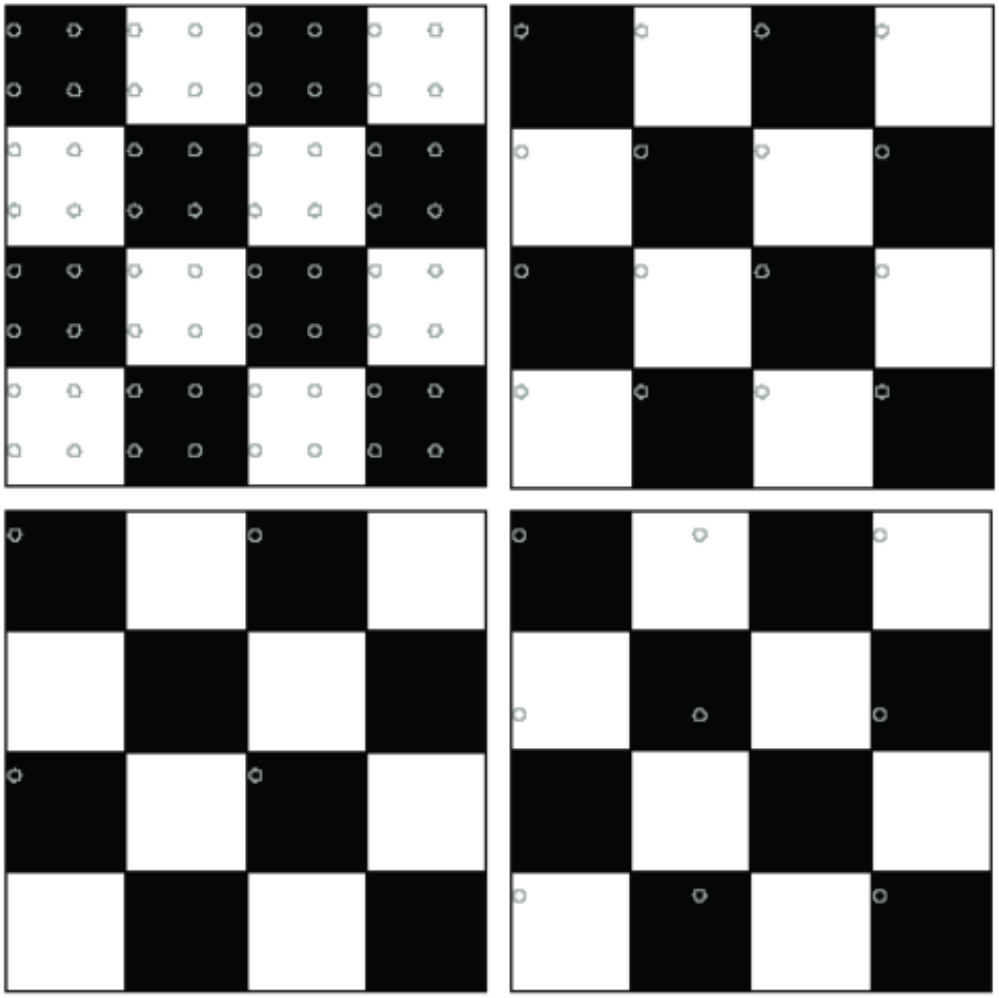
Aliasing

It happens when your sampling rate is not high enough to capture the amount of detail change.



To do sampling right, you need to understand the structure of your signal/image
The minimum sampling rate is called the Nyquist rate:
at least twice the signal frequency

2D example



Good sampling

Bad sampling

[Source: N. Snavely]

Down sampling

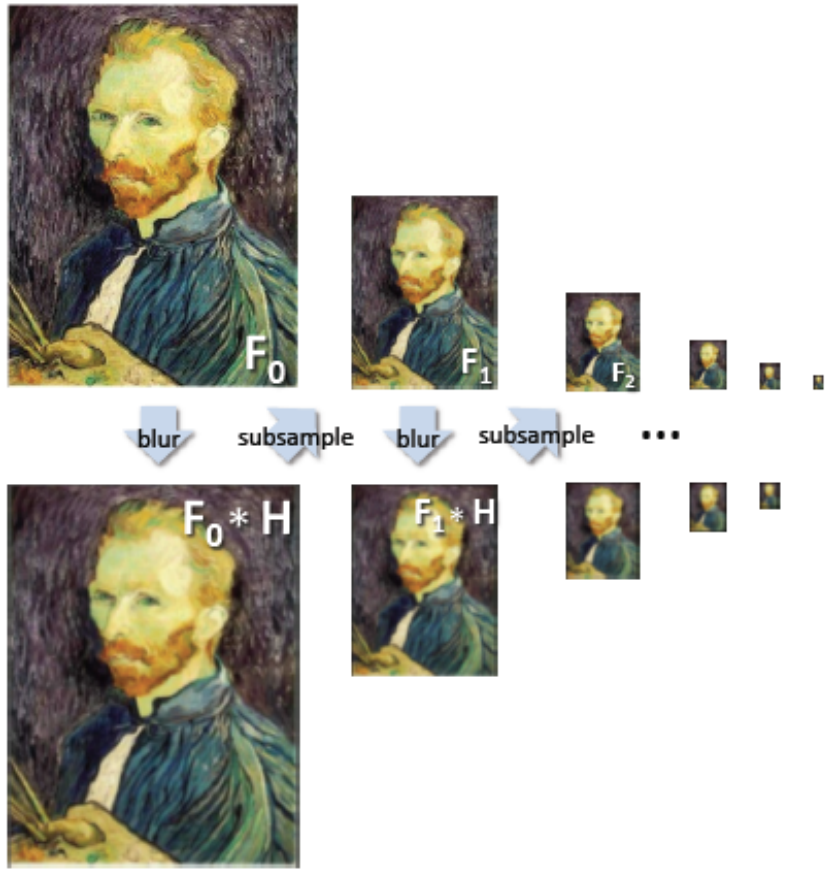
High frequencies are due to edges

Downsampling by a factor of two is two low for the sharp edges

What can we do?

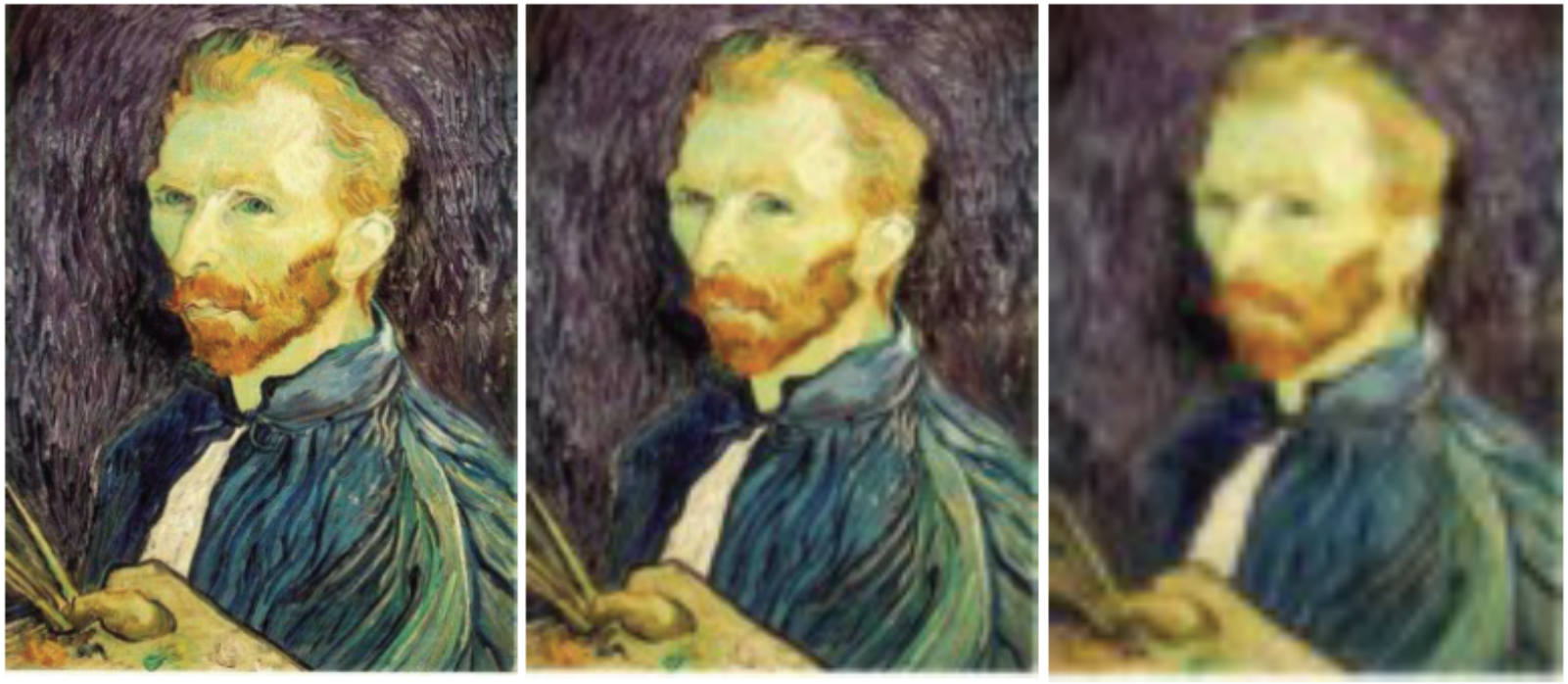
Gaussian pre-filtering

Solution: blur the image using a Gaussian filter, before down-sampling



[Source: N. Snavely]

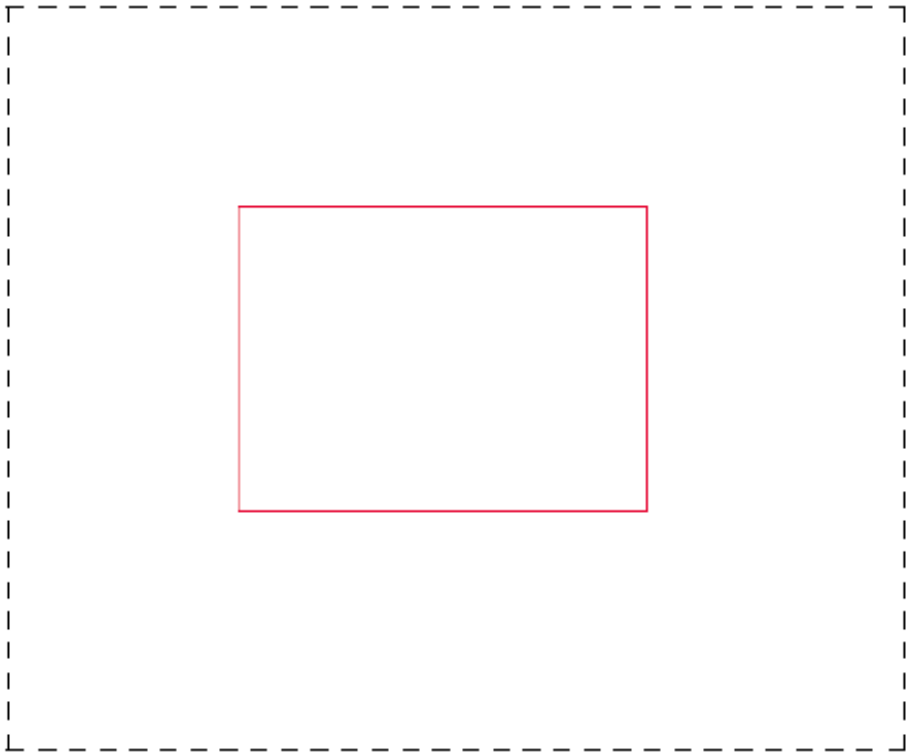
Sub-sampling with Gaussian pre-filtering



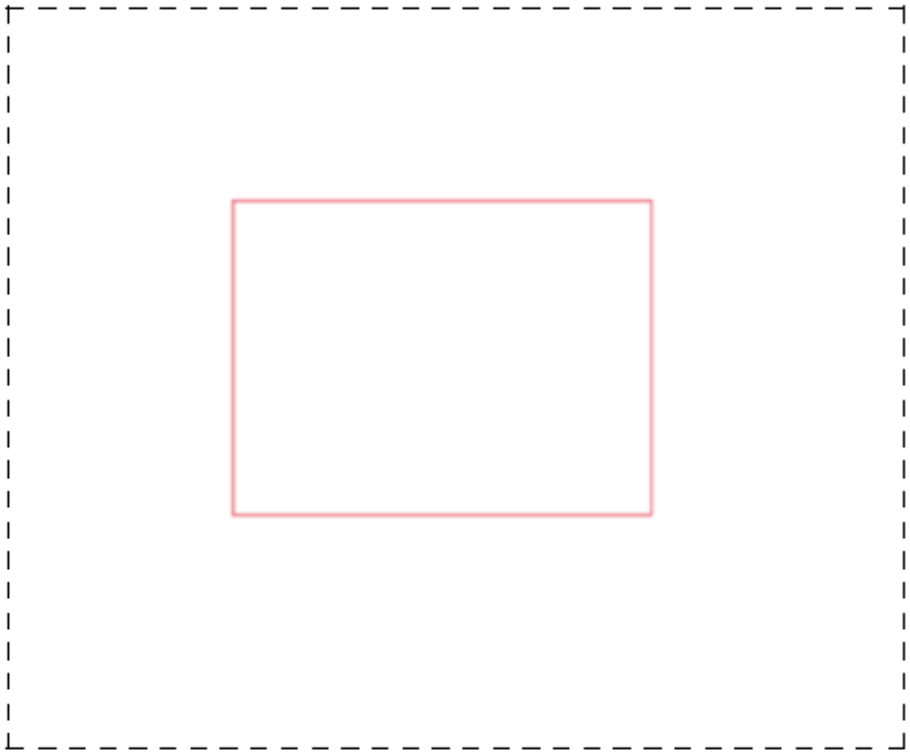
Gaussian 1/2 G 1/4 G 1/8

[Source: S. Seitz]

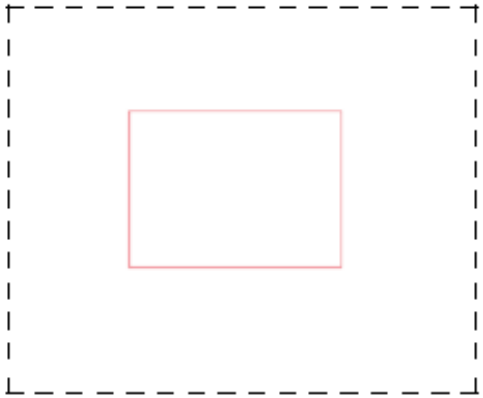
Where is the rectangle?



Where is the rectangle?



Where is the rectangle?



Where is the chicken

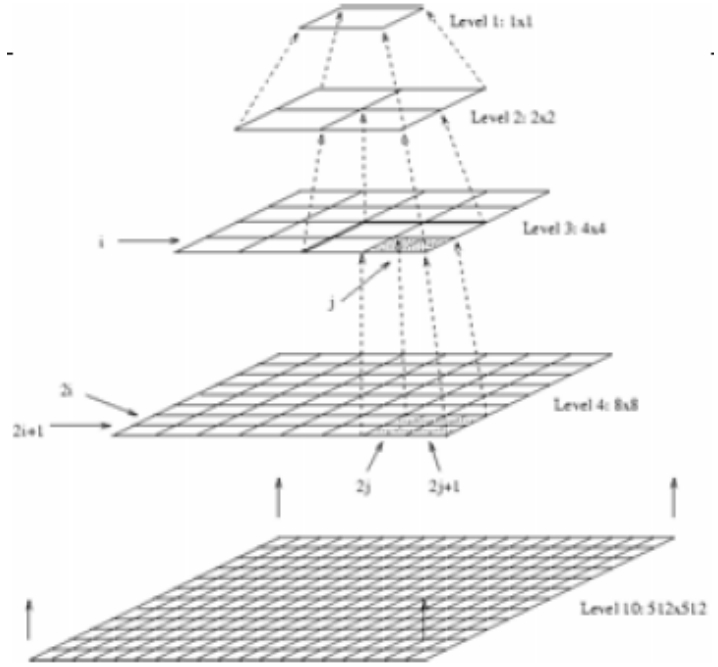
Where is the chicken

Where is the chicken

Gaussian Pyramids

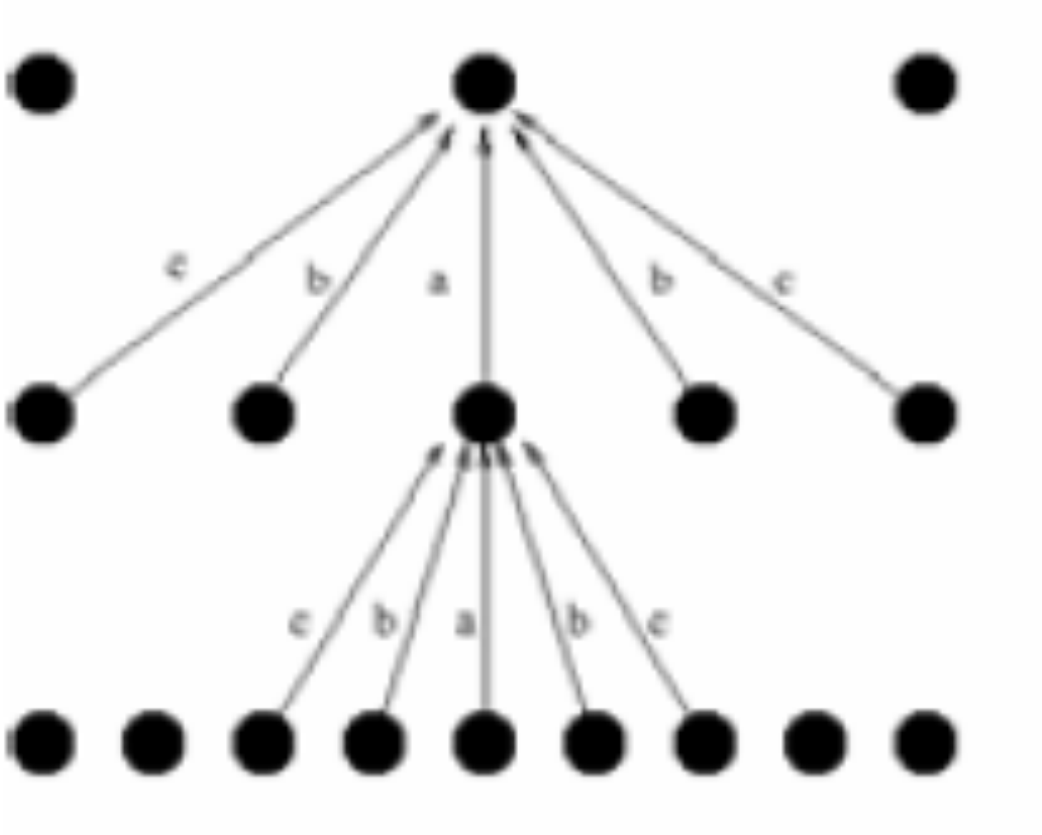
Next level is obtained by smoothing the previous level:

$$I_l = \text{reduce}(I_{l+1})$$

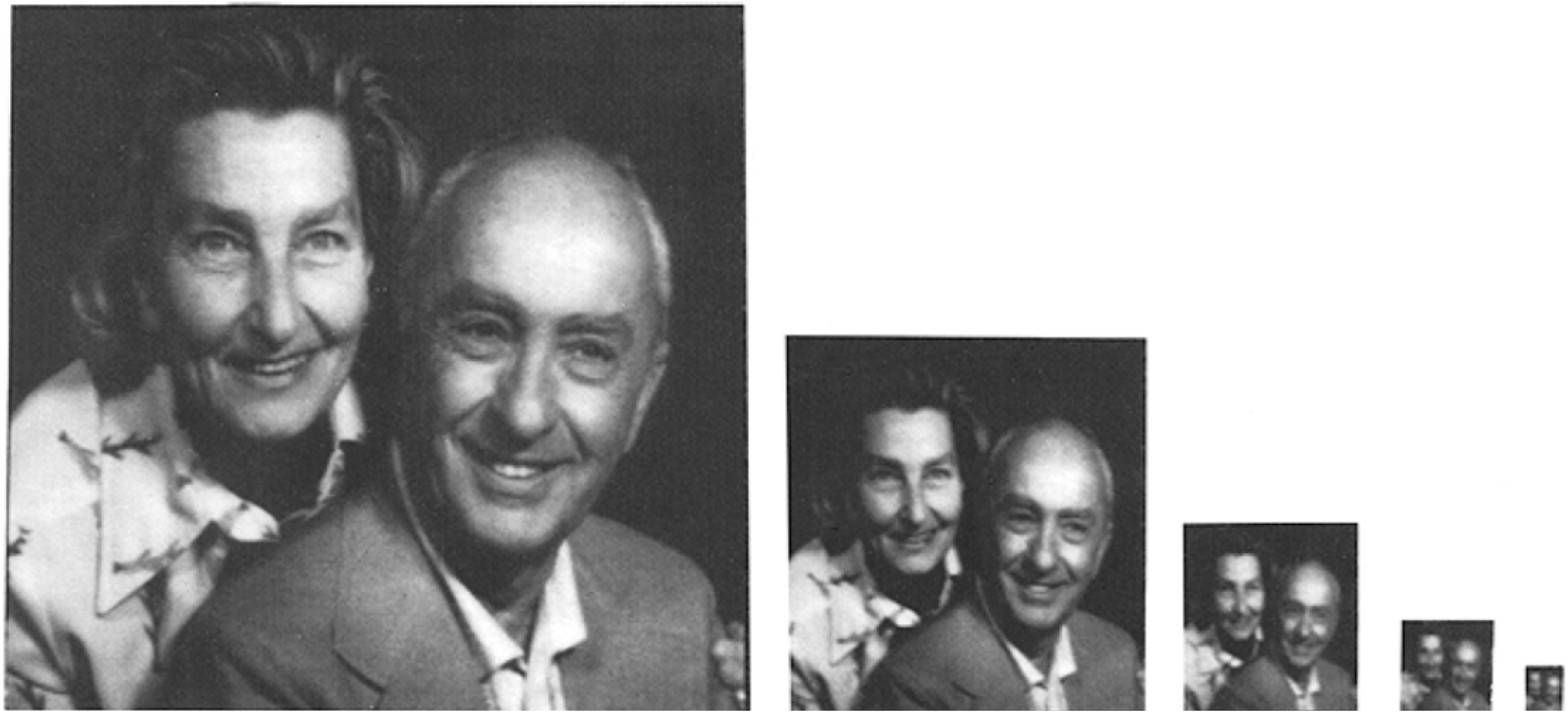


$$I_l(i,j) = \sum_{m=-2}^{m=2} \sum_{n=-2}^{n=2} w(m,n) I_{l+1}(2i+m, 2j+n)$$

Reduce



Reduce



Up-sampling

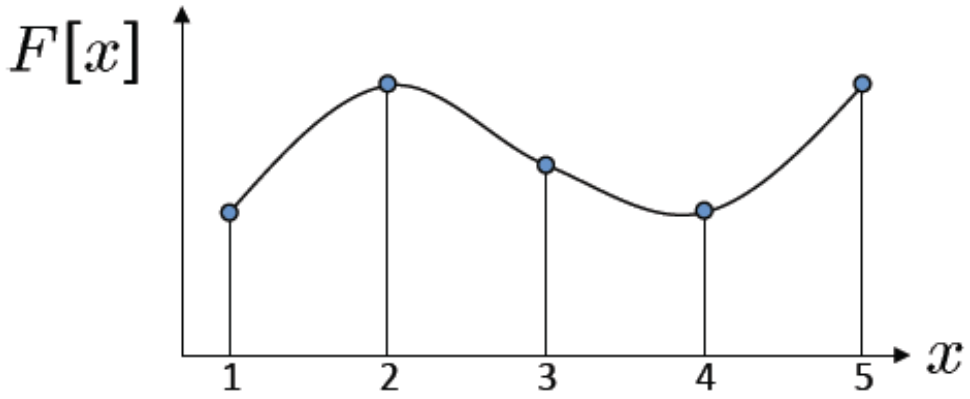
This image is too small. How can we make it 10 times bigger?

Up-sampling

This image is too small. How can we make it 10 times bigger?

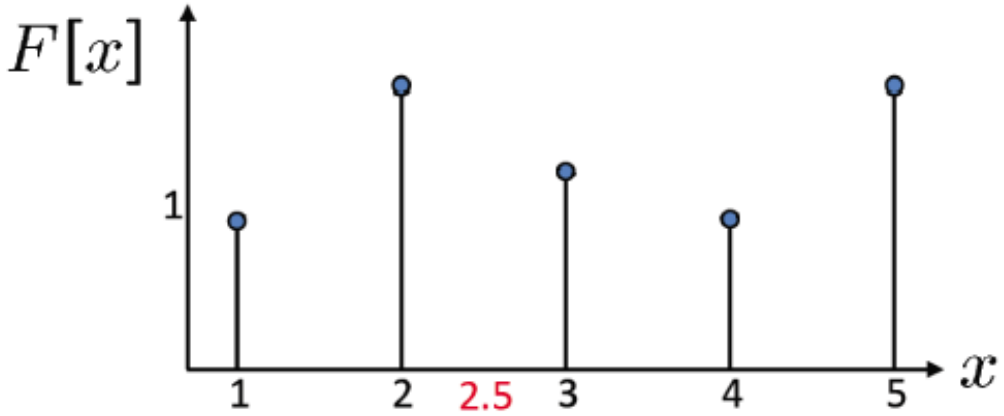
Repeat each row and column 10 times:

Interpolation



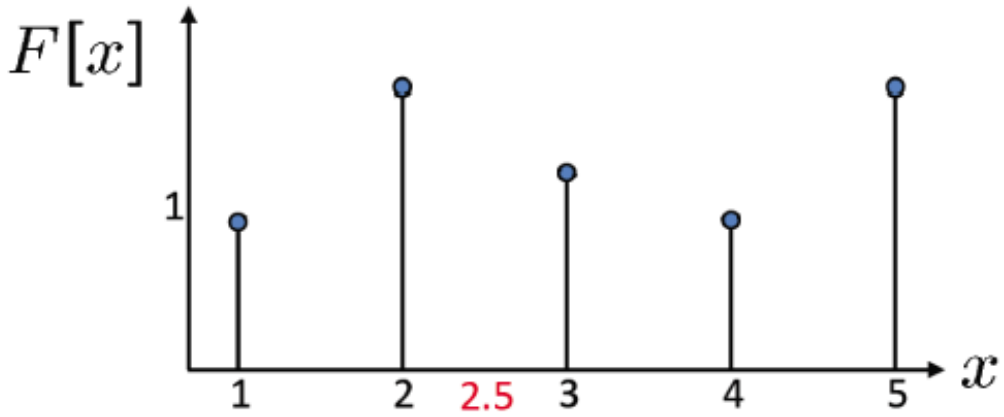
d = 1 in this example

Quantization: $F[x, y] = quantize\{f(xd, yd)\}$



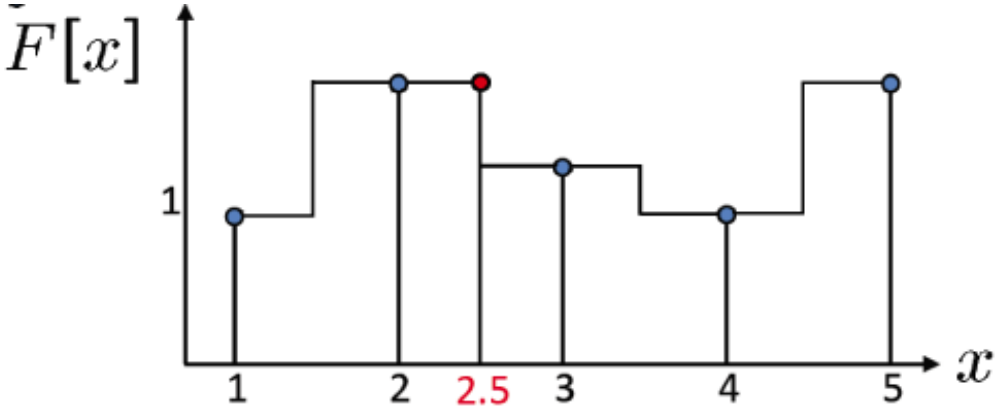
d = 1 in this example

Interpolation



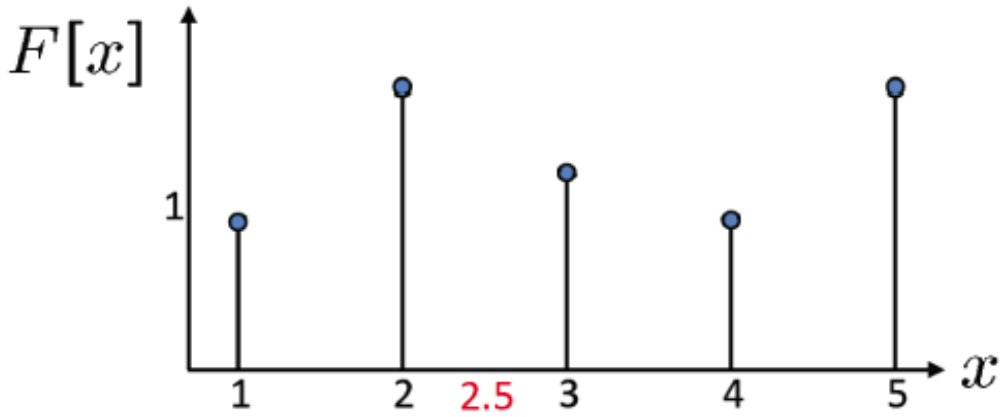
d = 1 in this example

Nearest Neighbor:



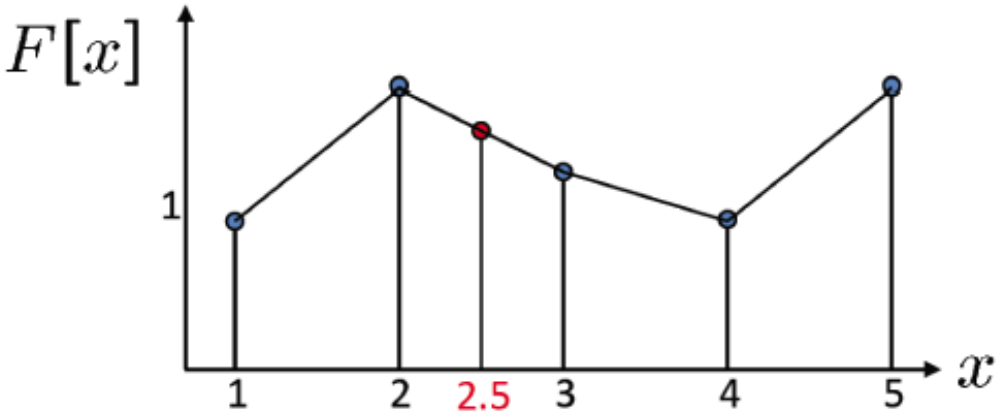
d = 1 in this example

Interpolation



d = 1 in this example

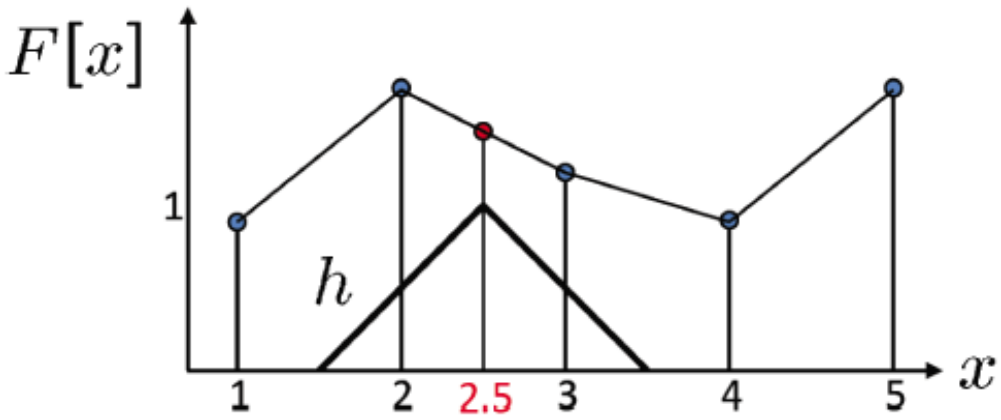
Linear:

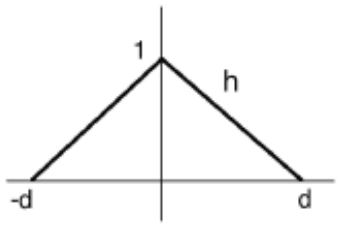


d = 1 in this example

$$G(x) = \frac{x_2 - x}{x_2 - x_1} F(x_1) + \frac{x - x_1}{x_2 - x_1} F(x_2)$$

Interpolation via Convolution





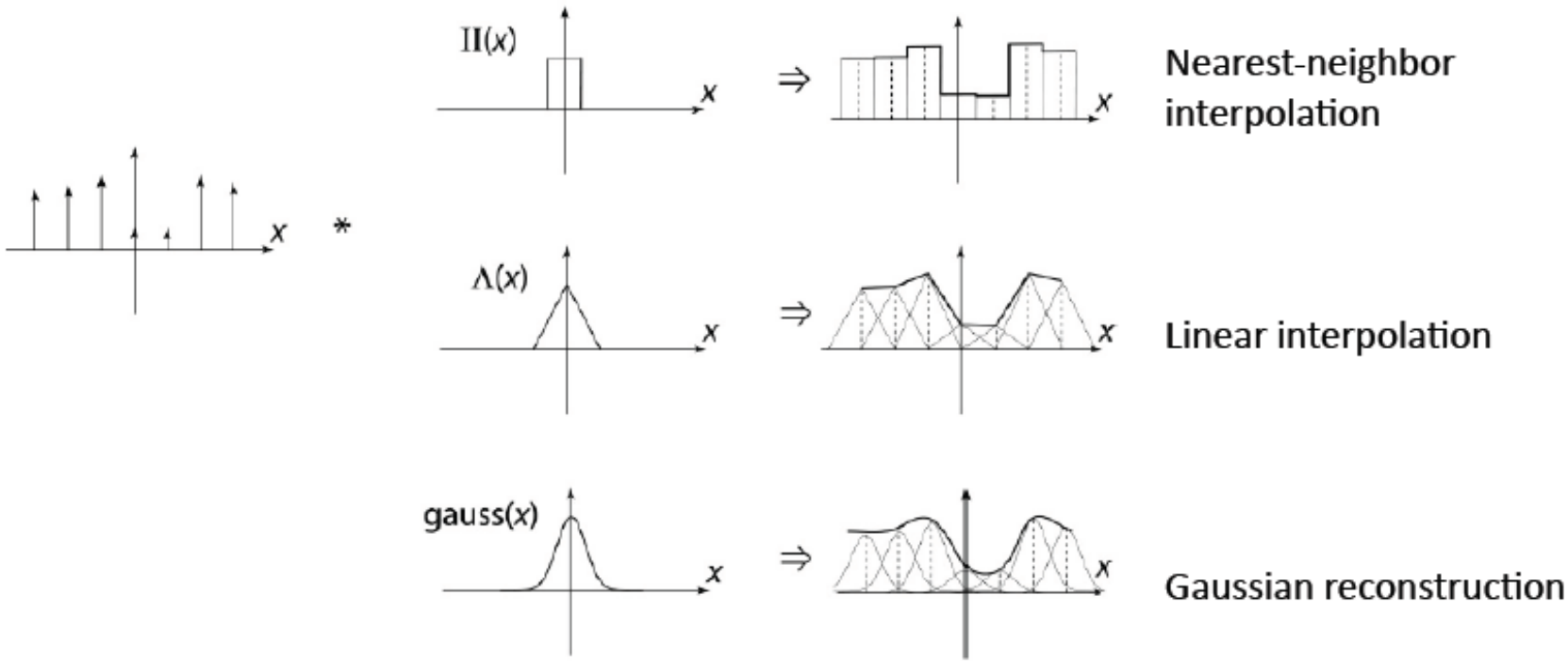
$$G(x) = \frac{x_2 - x}{x_2 - x_1} F(x_1) + \frac{x - x_1}{x_2 - x_1} F(x_2)$$

With $t = x - x_1$ and $d = x_2 - x_1$ we can get:

$$G(x) = \frac{d - t}{d} F(x - t) + \frac{t}{d} F(x + d - t)$$

$$G(x) = \sum_t h(t) F(x - t)$$

Interpolation via Convolution

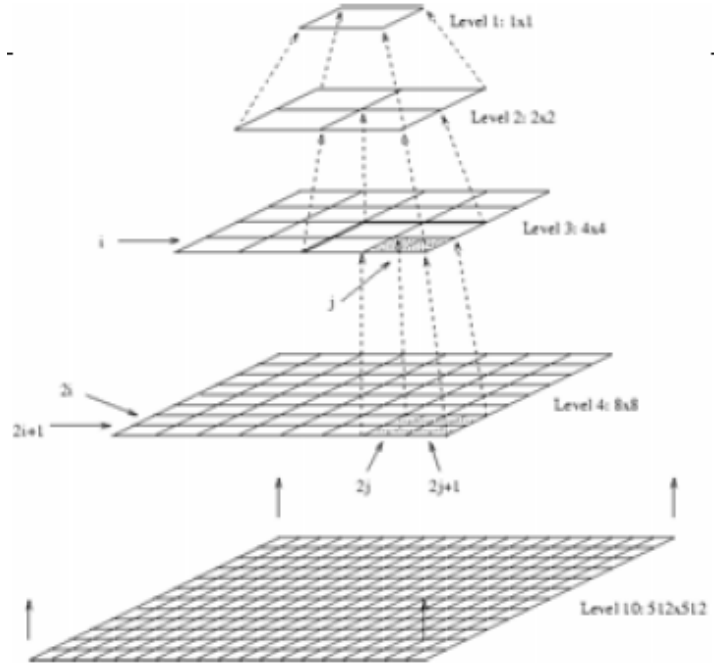


Source: B. Curless

Gaussian Pyramids

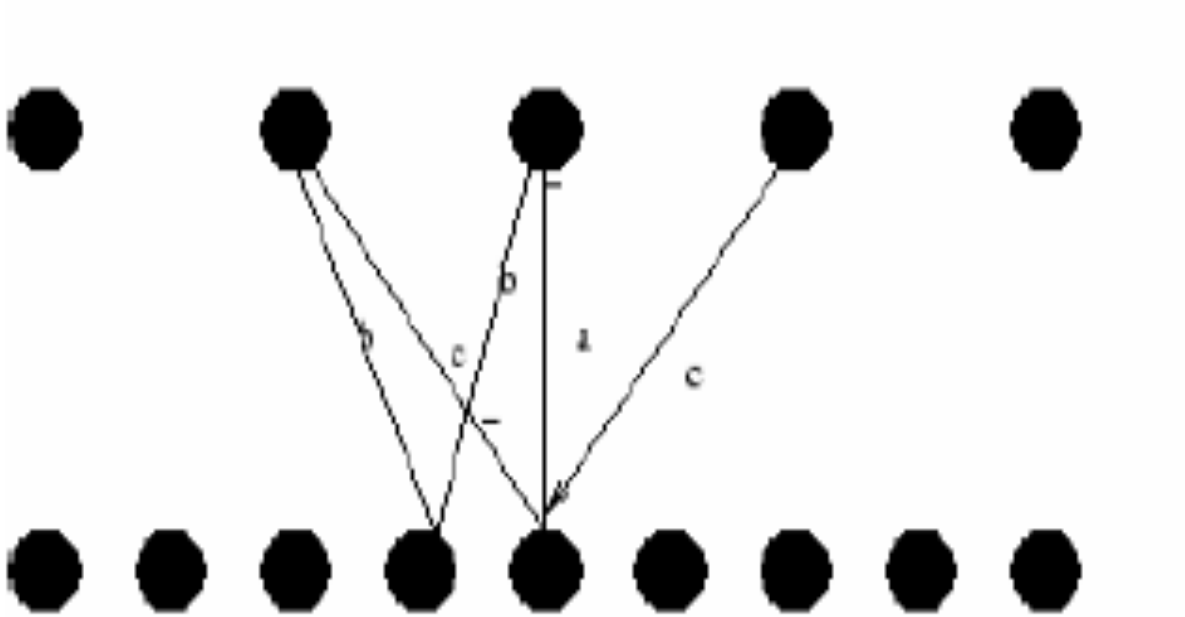
Previous level is obtained by interpolating:

$$I_{l-1} = \text{expand}(I_l)$$



$$\hat{I}_{l-1}(i,j) = \sum_{m=-2}^{m=2} \sum_{n=-2}^{n=2} w(m,n) I_l(\frac{2i+m}{2}, \frac{2j+n}{2})$$

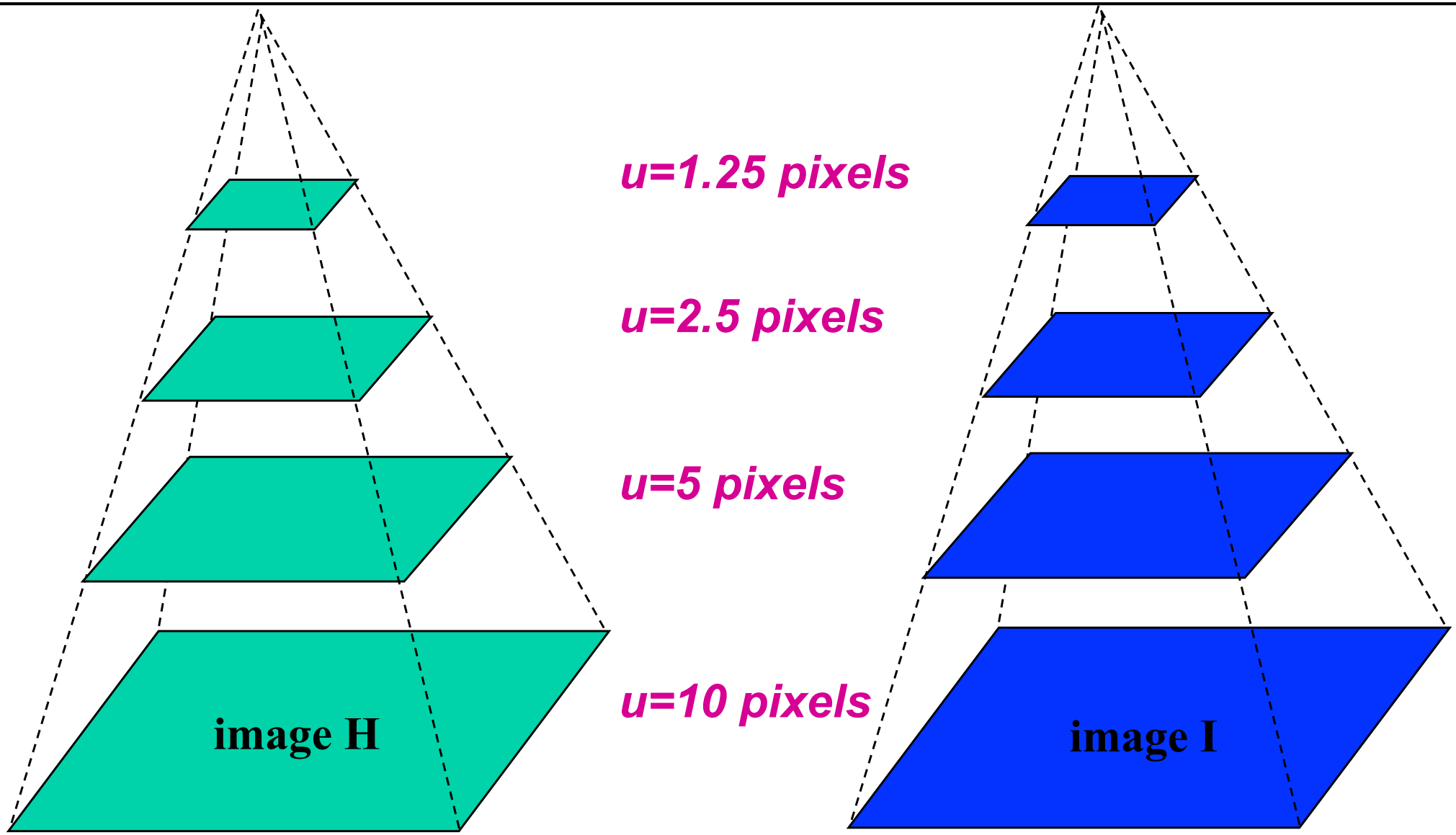
Expand



Expand

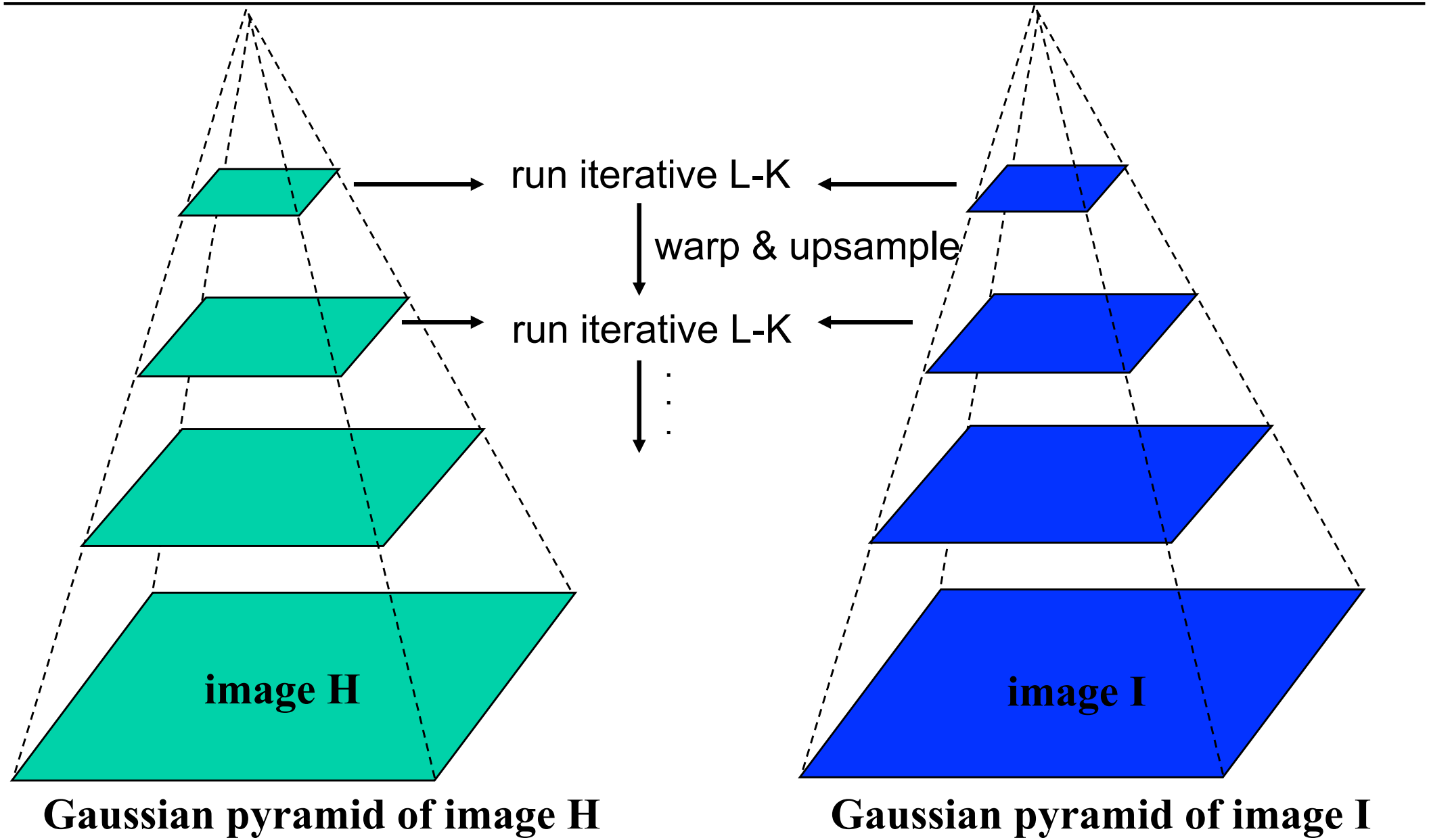


OF with Pyramids



Gaussian pyramid of image H **Gaussian pyramid of image I**

OF with Pyramids



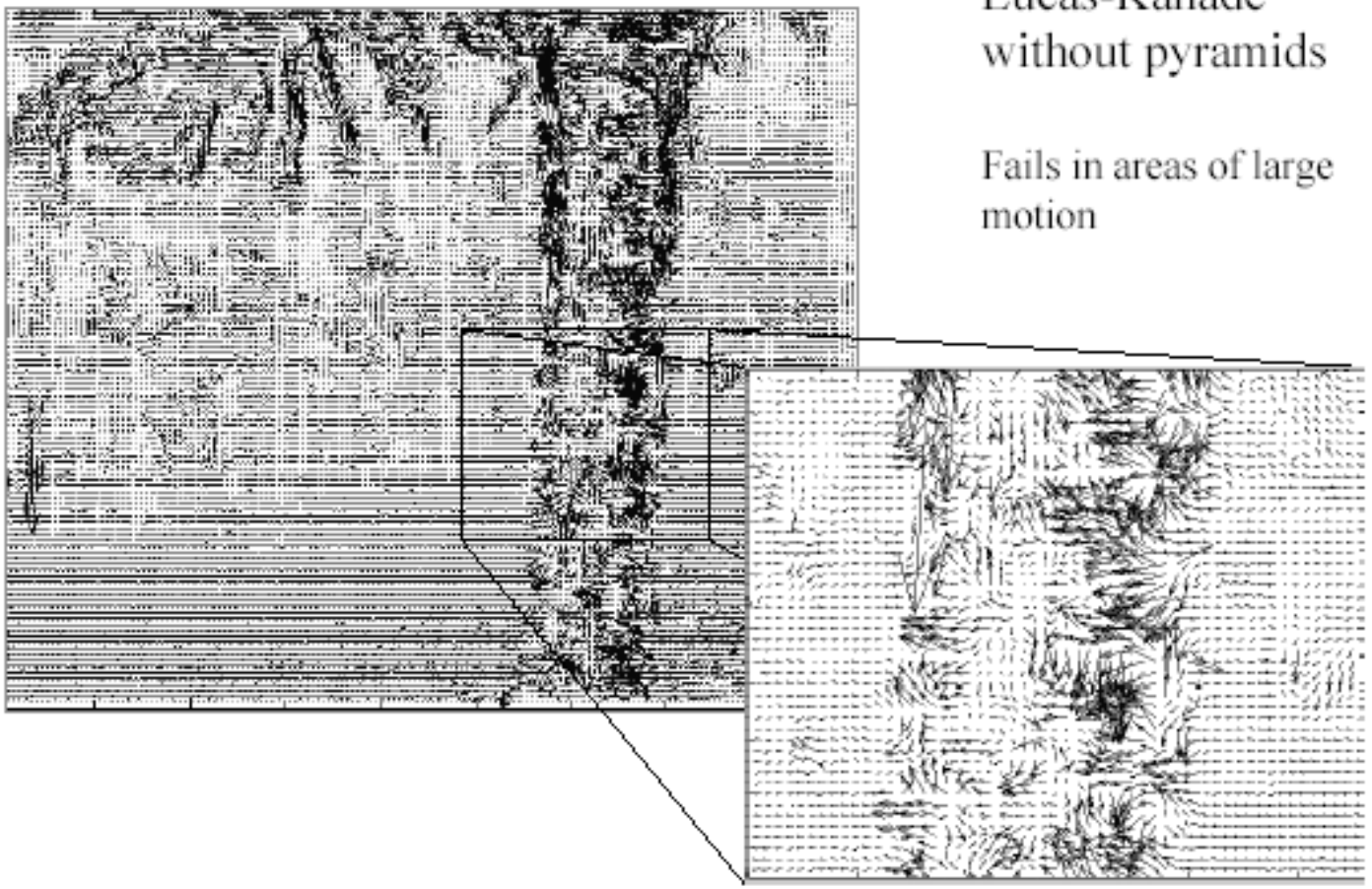
Gaussian pyramid of image H **Gaussian pyramid of image I**

Lucas Kanade with Pyramids

- Compute “simple” LK at lowest resolution
- At res. level i+1(higher resolution):

  - Take OF from level i
  - Expand by bilinear interpolation to create OF matrices of size of level i+1 (4 times as big)
  - Multiply OF components by 2
  - Compute I_t from a block displaced by OF
  - Apply LK OF to get correction
  - Add corrections to obtain OF at level i+1

OF without Pyramids



Lucas-Kanade without pyramids

Fails in areas of large motion

OF with Pyramids